How to use 'GIT'?

고명석 작성

GitHub : https://github.com/AlbertKo827

HomePage : http://myeongsku.com(공사중)

◉Git이란?

-'git'은 분산형 버전 관리 시스템'이다.

◉Git용어

- **리포지토리 (Repository)** : 말그대로 저장소이다.
- **작업 트리 (Working Tree)** : 현 작업자가 작업하고 있는 폴더

- **스테이징 영역** : 저장소에 커밋하기 전에 커밋을 준비하는 위치 (변경사항을 적용하기 전에 한번 더 변경사항을 정리하고 다듬을 수 있는 기회를 제공한다. 변경사항을 추가하기 위해서는 git add 를 사용한다. 커밋 예정인 변경사항이 있다고 보면 된다.)

-**커밋(Commit)** : 작업 트리에 변경된 내용을 저장하는 것이다. 커밋을 할 때는 해당 커밋에 관련하여 설명(Message)가 필수적으로 필요하다.

-**푸시(Push)** : 현재 원격 저장소 내의 해당 브랜치에 작업 트리의 변경 내용(커밋)을 업로드한다.

- **브런치(branch)** : 브랜치라는 것은 하나의 개발 라인을 의미 한다.
master 브런치를 기준으로 여러 브런치들이 상황에 맞게 분산이 된다.

(일종의 분류 사람을 만든다고 할 때, 사람에는 머리 , 팔, 다리 등 등이 있고, 머리에는 눈, 코, 입 등이 있다. 이런 경우 master로부터 '머리'라는 브런치를 만들고, '머리'로부터 '눈', '모', '입' 등의 브런치를 만드는 것이다.)

- **풀(Pull)** : 현재 로컬 저장소와 원격 저장소의 변경 내용(커밋)을 병합하여 새로 커밋한다.

- **머지(Merge)** : 병합작업이다. 현재 로컬 저장소에 다른 브랜치를 병합한다.
- **헤드(Head)** : 이러한 브랜치에는 HEAD(branch head)라는 것이 있는데 이는 한개의 브랜치 내에서 가장 최근에 커밋이 된 reference이다.
- **체크 아웃 (Checkout)** : 작업 트리를 저장소의 특정 브랜치와 일치 하도록 변경하는 작업(e.g. Master 브런치에서 Test 브런치로 변경)

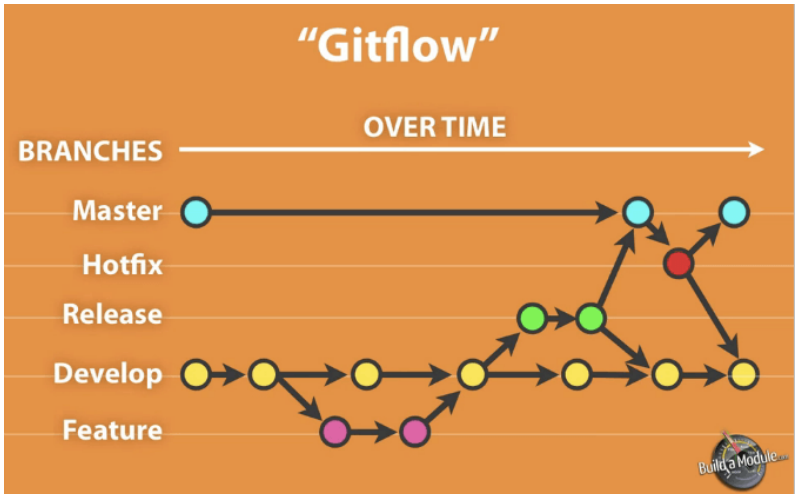
- **origin** : 보통 origin은 원격 저장소를 의미하며, 원격 저장소로부터 클론을 받을 때 기

본적으로 origin의 이름이 원격 저장소를 의미하게 된다. 만약, 다른 이름을 원한다면 origin이 아닌 다른 이름으로 원격저장소를 등록하면 된다.

@만약, 풀이나 머지 작업 중에 충돌이 발생하면 해당 충돌 부분을 인위적인 수정을 한 후 새로 커밋 후 푸시 해줘야 한다.

⦿Git Flow

- **Git Flow**는 Vincent Driessen의 브랜칭 모델을 위한 고수준 저장소 작업을 제공하는 git의 확장입니다.



⦿Git Flow 용어 및 설명

- Master : 가장 안정적인 브랜치이다. 이는 주로 배포버전이다.

- **Develop** : Master를 기반으로 존재하는 브랜치이다. 주 개발{패치(patch) 와 업그레이드(upgrade)}은 Develop 브랜치에서 이뤄지게 된다.

- **Feature** : 특정 기능을 개발을 할 때 Develop을 기반으로 만들어지는 브랜치이다. 예를 들면, Develop에서 얼굴을 개발 중일 때, '눈(eyes)'이라는 브랜치를 새로 만들어 개발을 진행하게 되는 것이다 (Branch Featrue/eyes). 개발이 완료된 후에는 Develop에 Merge된다.

- **Release** : Develop에서 버전 업을 위한 개발이 끝난 후 만들어지는 브랜치이다. Feature에서 '눈'의 기능이 개발이 끝나면 얼굴에 눈의 기능을 사용할 수 있게 되고, 이것은 새로운 버전이 된다 (Branch Release/0.1.0). 하지만 Release는 Master와 다르게 안정적이지 못한 상태이다. 즉, 새로운 기능과 함께 소수의 버그 또한 포함하고 있다. 이에 대한 어느정도 작업이 끝나면 Master와 Develop에 다시 Merge 된다.

- **Bugfix** : Release에서 발생한 버그를 수정할 때 만드는 Branch이다 (즉, Release 기반 Branch). 상황에 따라서 Bugfix를 만들거나 아니면 Release에서 즉각 수정을 한다.

- **Hotfix** : 배포 버전(Master)에서 발생한 긴급 버그를 수정할 때 만드는 Branch이다 (즉, Master 기반 Branch). Master 1.3.1 Version에서 버그가 발생하게 될 시 해당 버전 태그로 부터 Hotfix 브랜치를 따고, 그 안에서 버그를 수정한 후 Master와 Develop에 Merge되게 되며, 배포 버전은 1.3.2 Version이 되게 된다.

◉SourceTree(Git GUI)

- SourceTree(GUI) : Git을 좀 더 간편하게 사용할 수 있도록 해주는 GUI 툴이다.

Git을 좀 더 편하고 쉽게 사용할 수 있으며, Git Flow 기능도 포함되어있어 간편하고 효율적인 버전 관리가 가능하다.

◉How to use Git with GitFlow

※Git과 Git Flow 설치 (Git Flow는 windows 기준으로 설치방법을 추가 했습니다.)

#Git install

https://git-scm.com/에서 git을 설치해 줍니다.

#GitFlow install

CMD를 열고

- git clone –recursive git://github.com/nvie/gitflow.git

명령으로 gitflow를 받아옵니다.

- getopt.exe // http://sourceforge.net/projects/gnuwin32/files/util-linux/2.14.1/util-linux-ng-2.14.1-bin.zip/download?use_mirror=jaist
- libintl3.dll // http://sourceforge.net/projects/gnuwin32/files/libintl/0.14.4/libintl-0.14.4-bin.zip/download?use_mirror=jaist&download=
- libiconv2.dll // http://sourceforge.net/projects/gnuwin32/files/libiconv/1.9.2-1/libiconv-1.9.2-1-bin.zip/download?use_mirror=jaist&download=

그 다음 git 설치 폴더 내에 bin폴더에 위의 파일(해당 URL에서 다운 받은 후 압축을 푼 bin 폴더 내에 있음)들을 넣어준다.

※CMD를 이용한 사용법(Git)

우선 git을 사용할 프로젝트 폴더로 이동을 한다.

- cd "Project Directory"

이동한 후 Git을 Initialize 해준다.

- git init

이제 해당 폴더는 Local Repository가 됐다. 폴더 내 파일들을 스테이지에 올려준다.

-git add . //현 폴더 내에 모든 파일들을 스테이지에 추가하여 git이 추적할 수 있도록 한다.

이제 스테이지에 올라간 파일들이 변경이 되면 git에서 감지할 수 있게 된다.

파일들을 수정한 후에 커밋을 시켜준다.

-git commit –m "Message"

커밋을 할 때는 무엇에 대한 커밋인지 꼭 메시지를 추가하여야 한다.

이제 원격 저장소(Remote Repository)를 추가해준다.

-git remote add "저장소 단축 이름(보통 origin 사용)" "원격 저장소(Remote Repository) URL"

이제 푸시를 해주게 되면 로컬 저장소의 HEAD가 원격 저장소에 올라가게 된다.

-git push –u "저장소단축이름(origin)" "branch(master)"

'-u'속성은 '—set-upstream' 속성의 약자로 처음 푸시할때만 사용해주면 된다.

만약 다른 원격 저장소에서 클론을 생성을 하려면

- git clone "원격 저장소(Remote Repository)" "사용할 로컬 저장소(Local Directory)"

명령을 통해 로컬 저장소 경로에 해당 프로젝트를 저장해준다.

Clone 을 통해서 가져온 프로젝트는 'origin'을 '원격 저장소 단축 이름으로 사용'한다

※CMD를 이용한 사용법(GitFlow)

clone이나 새로 지정한 Git Local Repository로 CMD의 경로를 이동해준다.

그 후, GitFlow를 Initialize 해준다.

- git flow init

그 후에 Master, Develop, Release, Feature, Hotfix, Bugfix 등의 branch 폴더 이름을 성정해 주는 작업을 진행한다.

기본 값을 사용할 때에는 Enter를 쳐서 전부 넘겨주면 되지만 '-d'속성을 붙여주면 이런 작업 없이 바로 Initialize 한다.

- git flow init –d

작업이 끝나면 기본 master 브랜치만 있었지만 develop 브랜치도 추가됨과 동시에 checkout 된다.

특정 기능은 feature에서 개발을 진행하도록 한다. 'eyes'라는 기능을 개발을 시작하려면

- git flow feature start "Branch Name(eyes)"

명령을 치면 된다. 그 후에 'feature/branchName'의 브랜치가 생성됨과 동시에 checkout 된다. 'git flow feature start "eyes"' 라고 명령을 쳤다면 'feature/eyes'라는 브랜치가 생성되는 것이다.

해당 기능 브랜치를 원격 저장소에 올려 여러 개발자들과 같이 개발을 하고싶다면

- git flow feature publish "기능 브랜치 명"

명령을 통해 원격 저장소에 브랜치 추가가 가능하다.

'eyes'의 개발이 끝나면 develop에 Merge 작업을 해줘야 한다.

- git flow feature finish "Branch Name(eyes)"

명령을 치고 나면 develop으로 checkout됨과 동시에 'feature/branchName(eyes)'가

Merge 된다. 그 후 해당 브랜치는 제거된다. 'git flow feature finish "eyes"'라고 명령을 쳤다면 'feature/eyes'에서 develop으로 checkout되고 'eyes'가 Merge된 후 브랜치가 제거되는 모습을 볼 수 있다.

Release 역시 위와 마찬가지로

- git flow release start "Branch Name"

명령으로 시작을 한다. 다만, Release같은 경우 브랜치 명을 버전으로 표기해준다.

(e.g. "v0.2.0")

이 역시 다른 개발자들과 함께 버그를 수정하거나 해야한다면

- git flow release publish "릴리즈 브랜치 명"

명령을 통해 원격 저장소에 브랜치를 올릴 수 있다. 그리고 역시 끝나게 되면

- git flow release finish "Branch Name"

으로 끝내면 된다. Release 의 경우 Release의 브랜치 네임으로 태그를 넘긴다. 그 후 master에 Merge, develop에 merge 후 develop으로 checkout 된다. 이제 master로 checkout 후 원격 저장소에 push를 해준다. 이때, '--tags'속성을 붙여 원격 저장소에 해당 release 버전에 대한 태그를 남겨준다.

- git checkout master

- git push --tags "원격 저장소 단축 이름" master

이제 원격 저장소에 가보면 해당 태그들과 함께 새로 업데이트 된 저장소를 확인할 수 있다. 그리고 master를 push 해줄 때 develop도 같이 push 해준다.

- git push "원격 저장소 단축 이름" develop

이정도가 git에 대한 기본 사용법으로 볼 수 있다.

⦿How to use SourceTree GUI

SourceTree를 https://www.sourcetreeapp.com/에서 다운받아준 후, 설치한다.

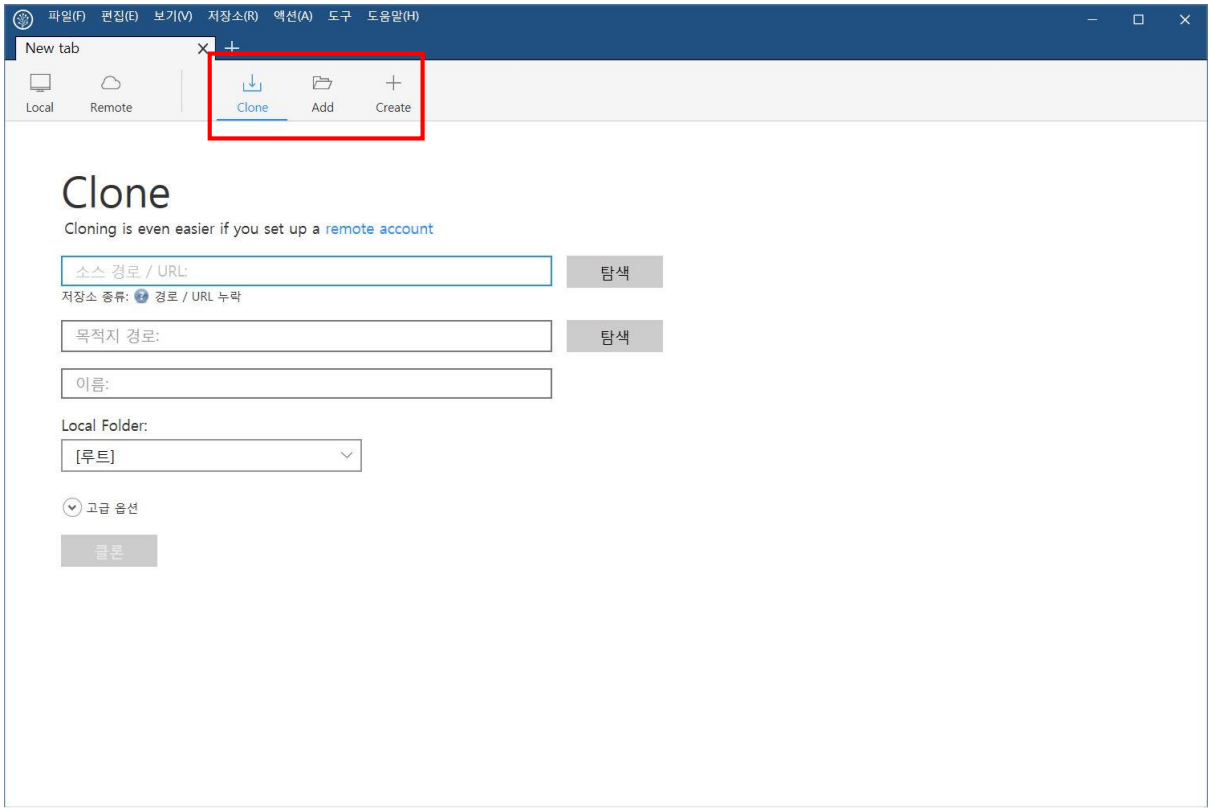
설치가 끝난 다음 SourceTree를 연다음 맨위 상단 탭에서

파일>복제/생성...

을 눌러 주거나

Ctrl + N

을 눌러 준다. 그러면 다음과 같은 창이 뜬다



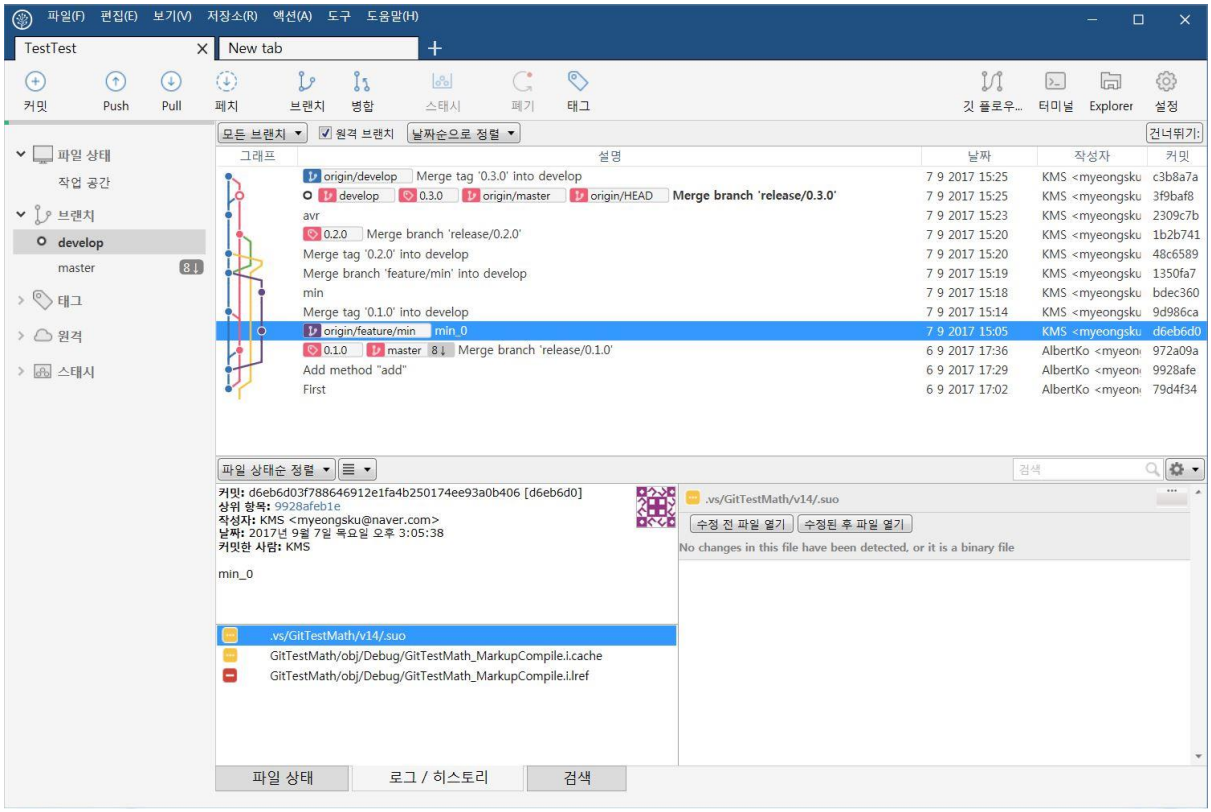
빨간 상자 안에 'Clone', 'Add', 'Create'가 있다.

#Clone : 로컬 / 원격 저장소로부터 프로젝트를 목적지 경로로 복사한다.

#Add : SourceTree에 기존에 있던 로컬 저장소를 불러온다.

#Create : 해당 폴더에 git을 initialize시킨다.

상황에 맞게끔 사용하면 된다.



위의 사진은 테스트용으로 만든 프로젝트를 SourceTree로 Clone으로 불러온 것이다.

CMD를 이용하여 사용하는 것 보다는 위의 SourceTree를 사용하는 것. 즉, Git GUI를 사용하는 것이 좀더 효율적이며 간편하다.

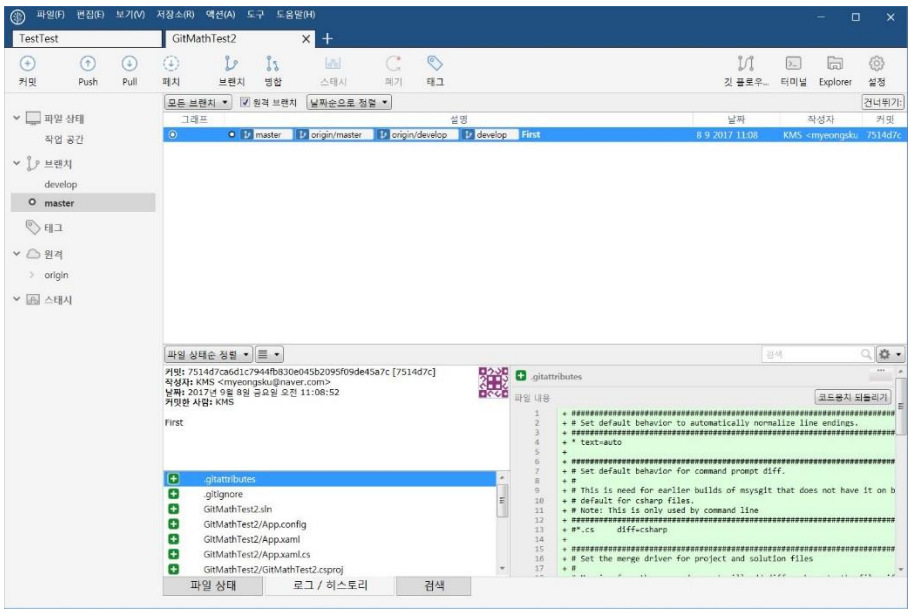
위의 '깃 플로우...'를 통해 GitFlow를 Initialize 할 수 있다.

'git flow init'과 같다고 보면된다.

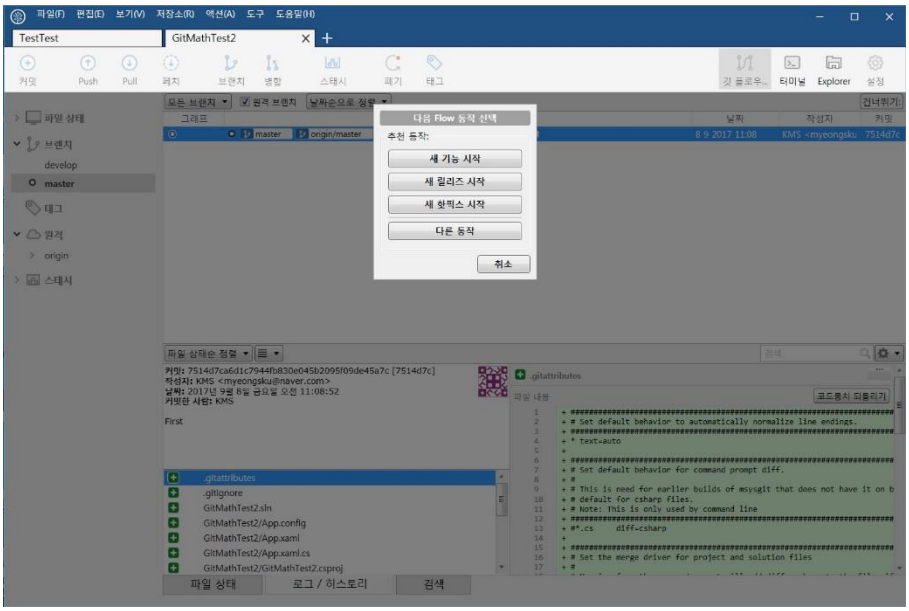
각 각 브랜치들의 이름들을 정의 해주고 나서 다시 '깃 플로우...'를 누르면

Feature, Release, Hotfix에 대한 작업을 진행 할 수 있다.

◉How to use SourceTree GUI (심화)



위의 사진은 현재 내 컴퓨터에서 SourceTree를 이용해 Local Git Repository를 만든 후 그 안에 WPF Math 프로젝트를 만든 후 Git Flow까지 Initialize 시킨 SourceTree의 사진이다.

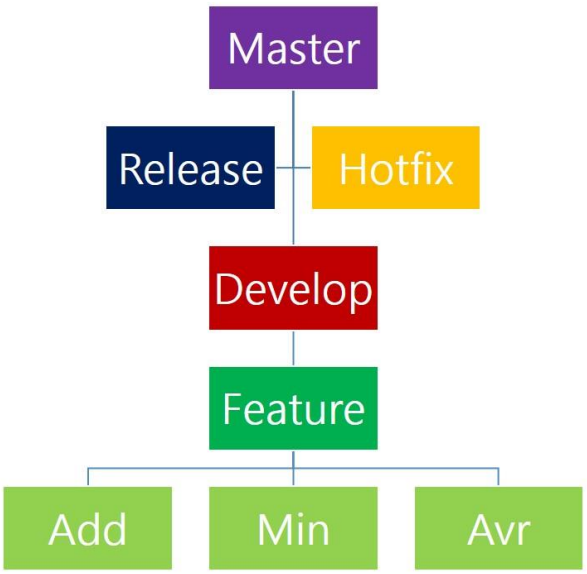


Git Flow를 Init 하고 난후 위의 '깃 플로우...'을 누르면 다음과 같은 창이 뜬다.

- 새 기능 시작 (Start new feature) =>git flow feature start
- 새 릴리즈 시작 (Start new release) =>git flow release start
- 새 핫픽스 시작 (Start new hotfix) =>git flow hotfix start

'다른 동작'은 만약 기존에 이미 진행을 하던 작업들이 있으면 그에 대한 동작으로,

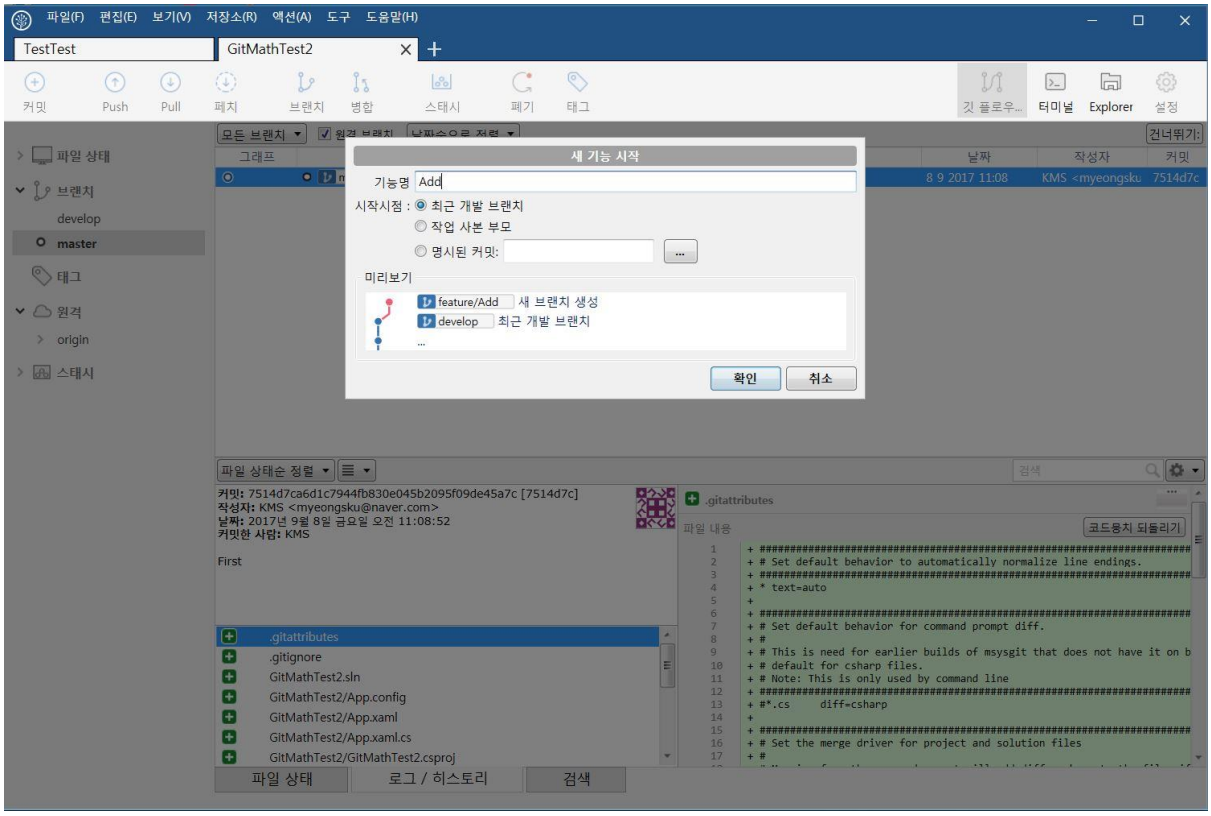
- 기능 마무리 (Finish feature) =>git flow feature finish
- 릴리즈 마무리 (Finish release) =>git flow release finish
- 핫픽스 마무리 (Finish hotfix) =>git flow hotfix finish



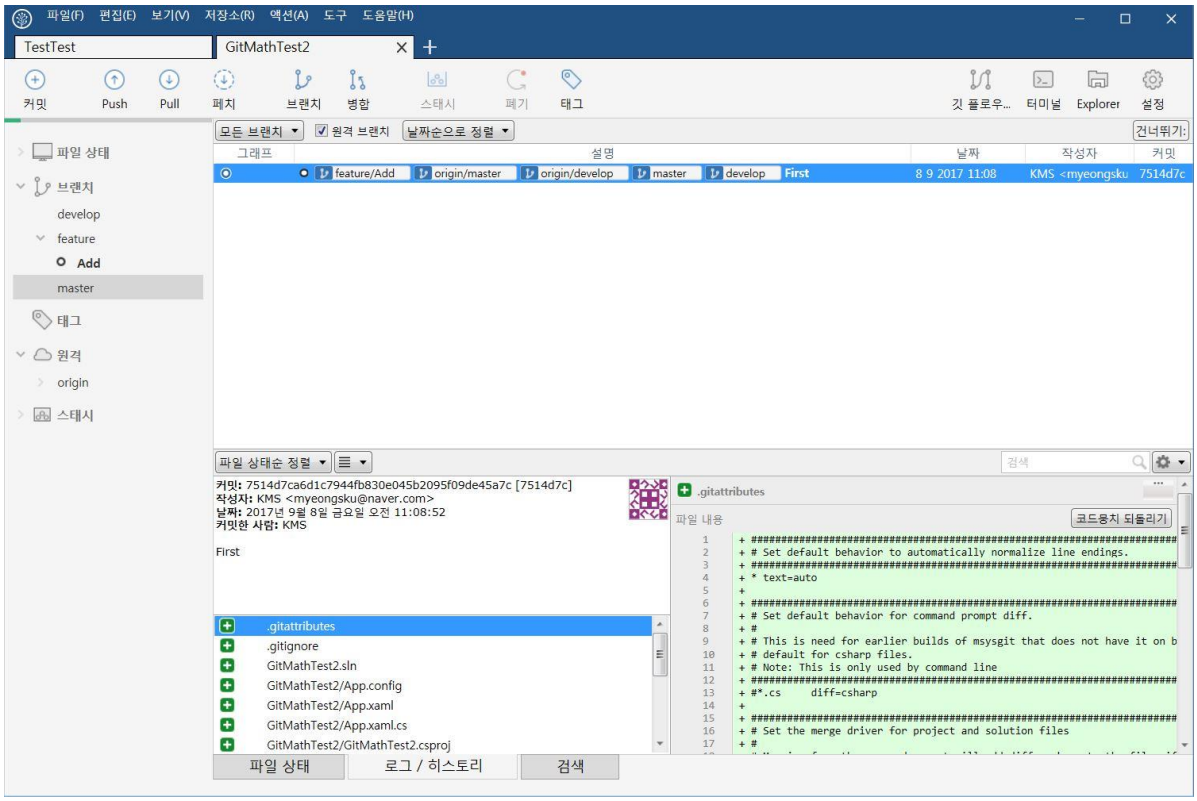
이번에 GitFlow를 이용해 위의 모델을 구현 후 Remote Repository로 Push하는 것까지 작업을 진행을 할 것이다. 기능 부분에 'Add', 'Min', 'Avr'는 각각 서로 다른 컴퓨터로 구현을 했다.

작업 순서는 대략적으로

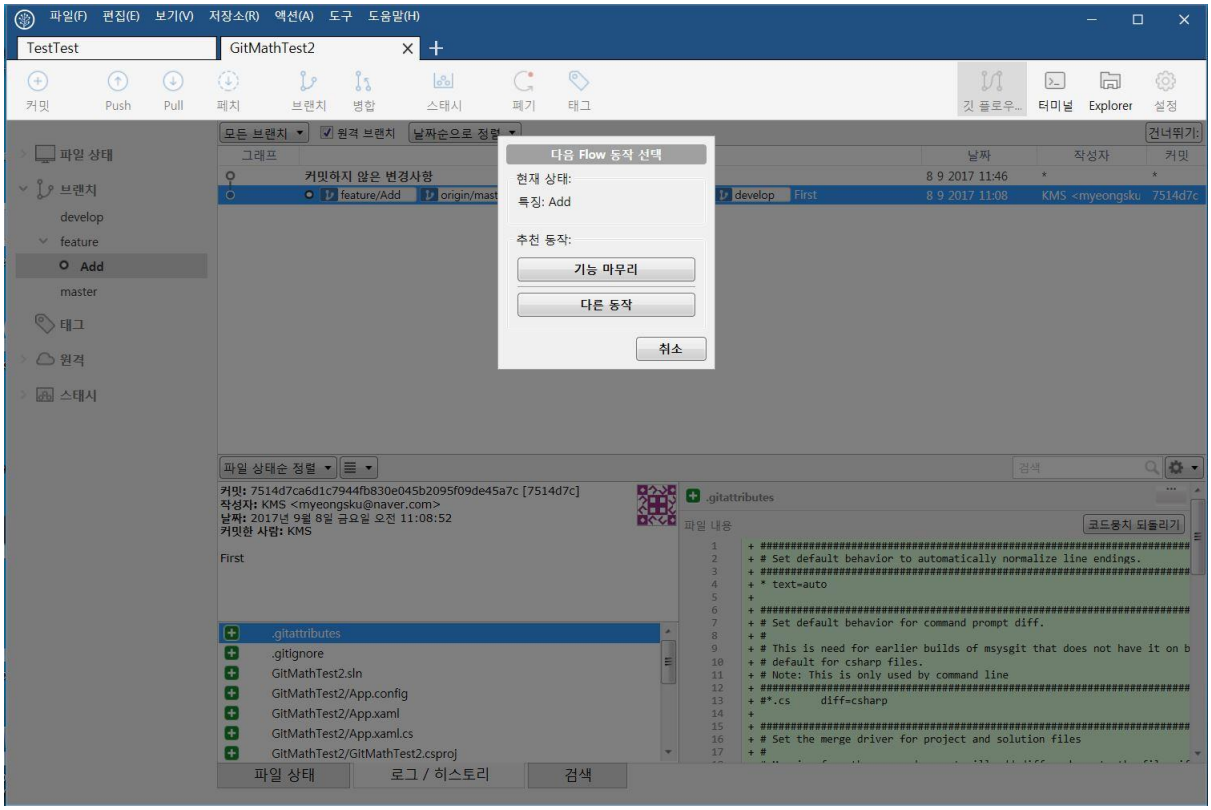
(User1) Start Feature/Add / (User2) Start Feature/Min / (User3) Start Feature/Avr
=> (User1) Finish Feature/Add 후 origin/develop에 push => (User1) Start Release 0.1.0
=> (User2) Finish Feature/Min & develop pull 후 origin/develop에 push
=> (User1) Finish Release 0.1.0 후 origin/develop & origin/master에 push
=> (User3) Finish Feature/Avr 후 develop pull 후 origin/develop 에 push
=> (User3) Start Release 0.2.0 => Master로부터 Start Hotfix 0.1.1 => Finish Hotfix 0.1.1
=> origin/develop & origin/master에 push
=> (User3) Finish Release 0.2.0 후 origin/develop & origin/master에 push



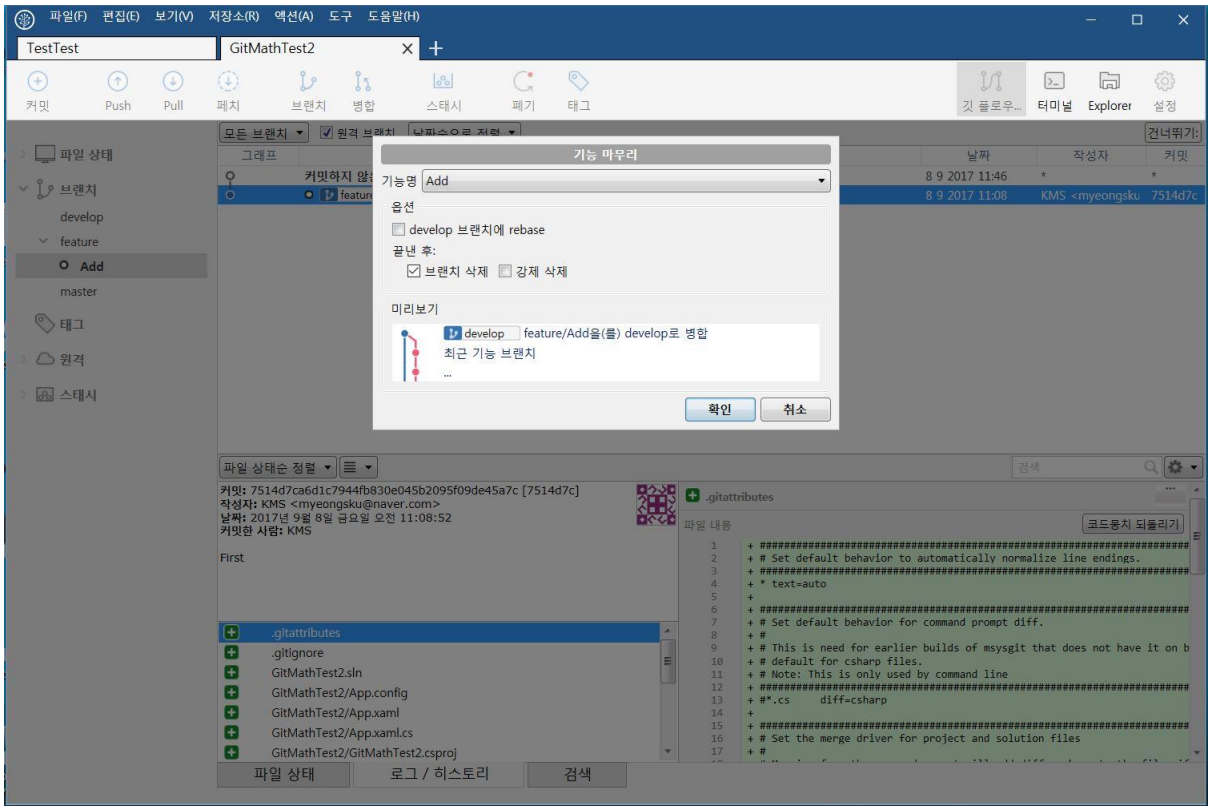
우선 Feature/Add를 추가 해준다. (같은 방법으로 다른 User들도 각각 Feature/Min, Feature/Avr을 추가 해준다.)



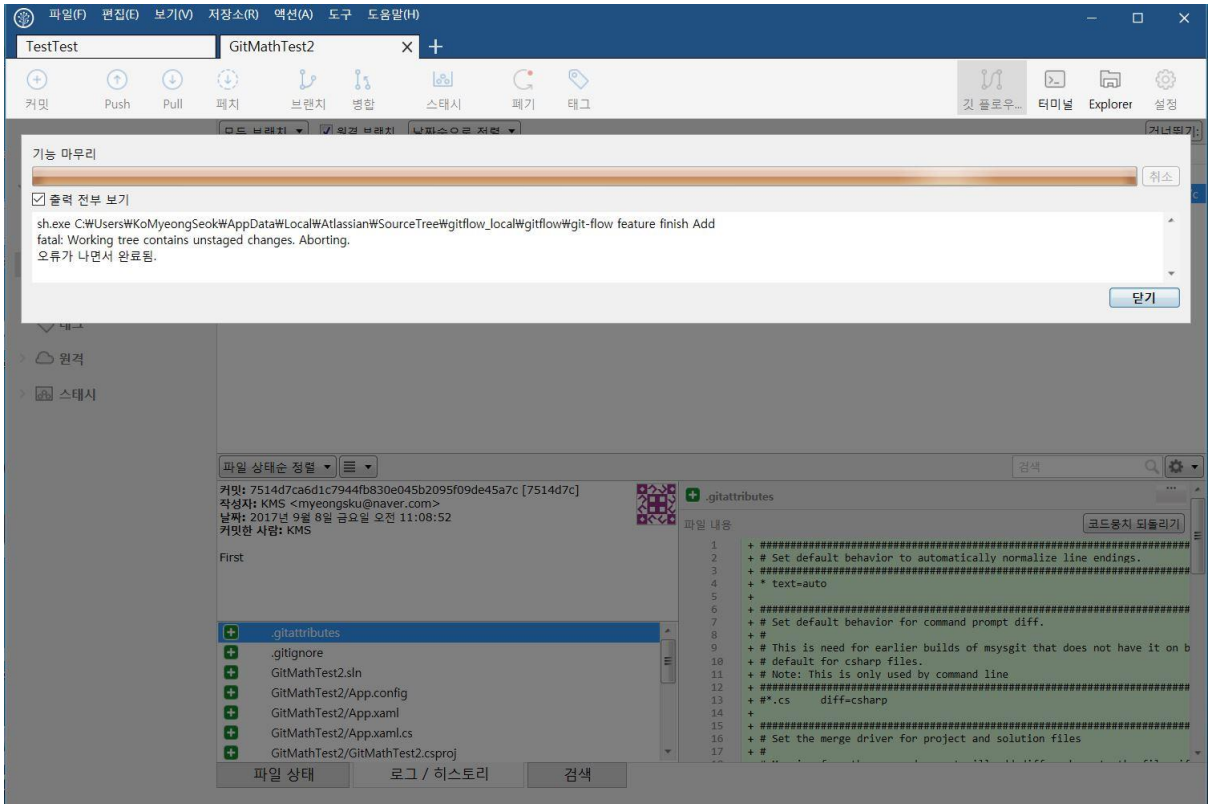
Feature/Add를 추가해주고 난 후의 모습이다.



기능 마무리 (git flow feature finish "add")를 눌러주면 Feature/Add를 Local 상의 develop 브랜치에 Merge 후 제거 되게된다.

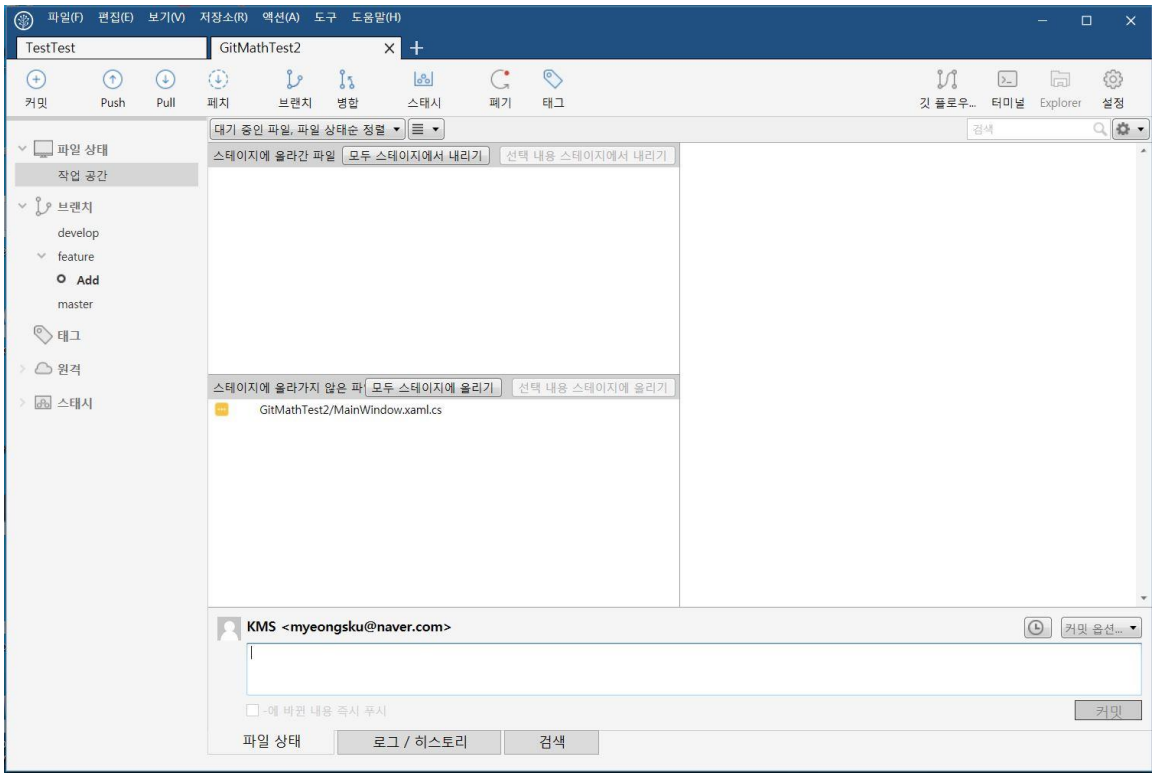


확인을 눌러준다.

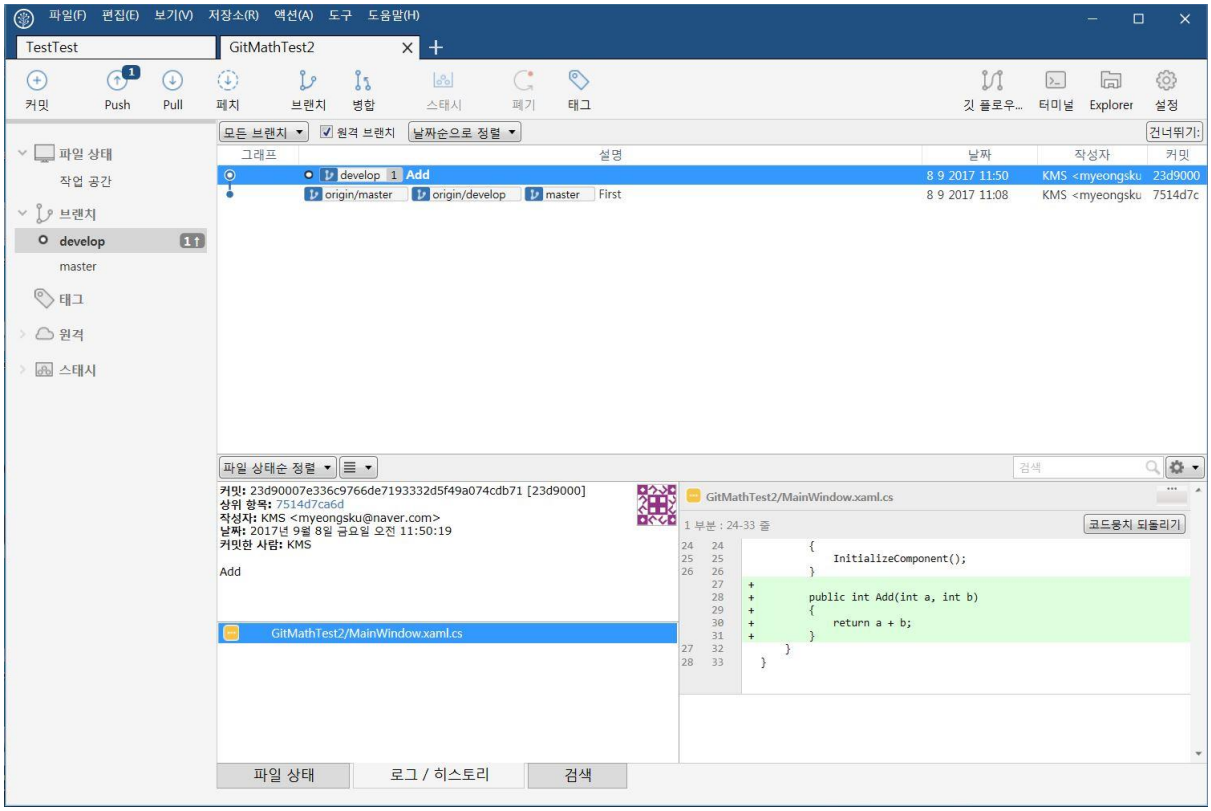


확인을 눌렀더니 위와 같은 에러가 나면서 작업이 진행이 되지않는다.

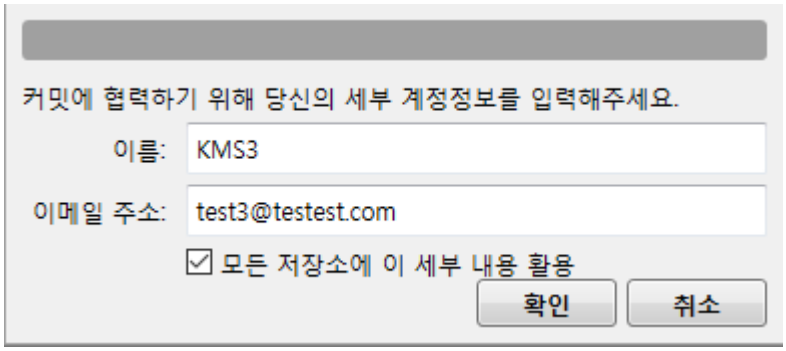
이유는 현재 스테이지(index)에 파일들을 추가 시켜주지 않았기 때문이다.



모두 스테이지에 올리기를 누른 후 변경 내용을 타이핑 후 커밋을 해준다.

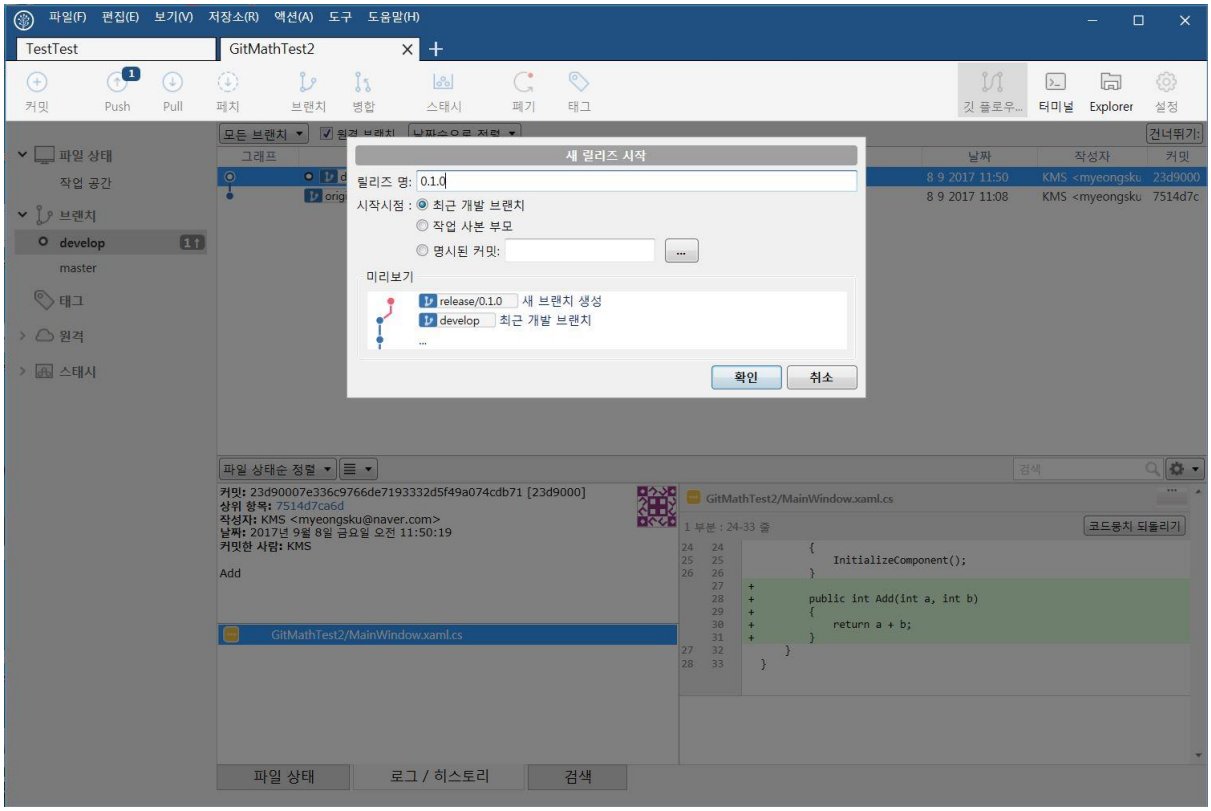


develop 브랜치에 Feature/Add가 성공적으로 Merge가 되었다.

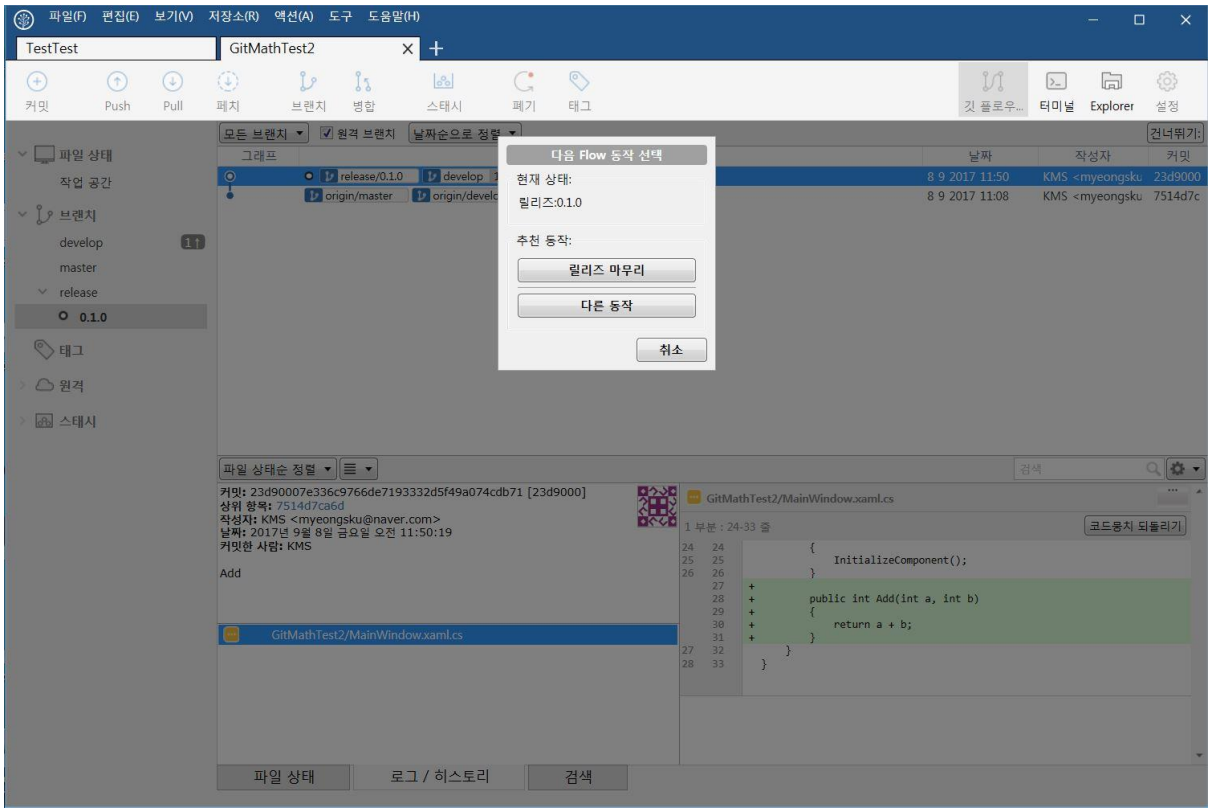


만약, git을 설치한 후 git config user.name 'name' 과 git config user.email 'email'을 설정해 주지 않았다면 다음과 같은 창이 뜬다. 자신의 정보를 입력해 주면 된다.

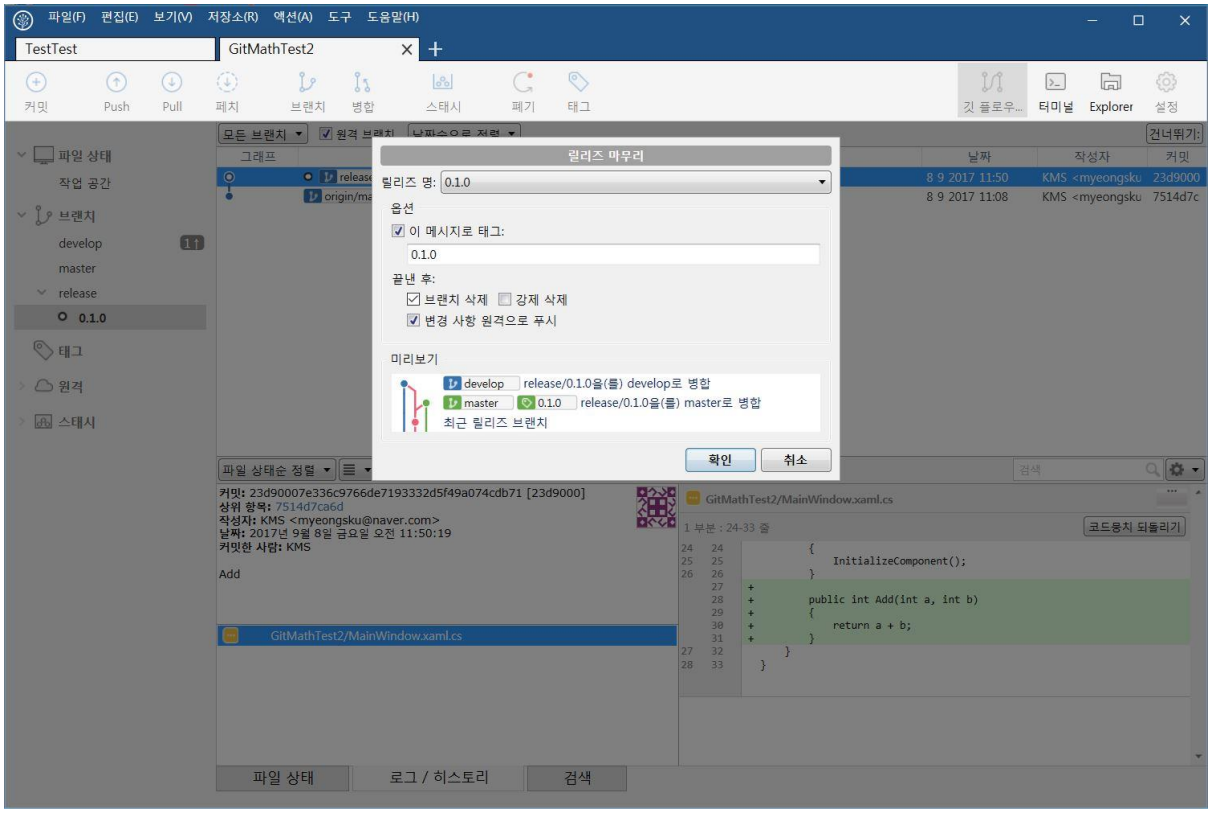
계정을 등록해주면 하단에 사용자 정보가 뜬다. 커밋을 하거나 태그를 달아주면 위의 계정 정보들로 등록이 된다.

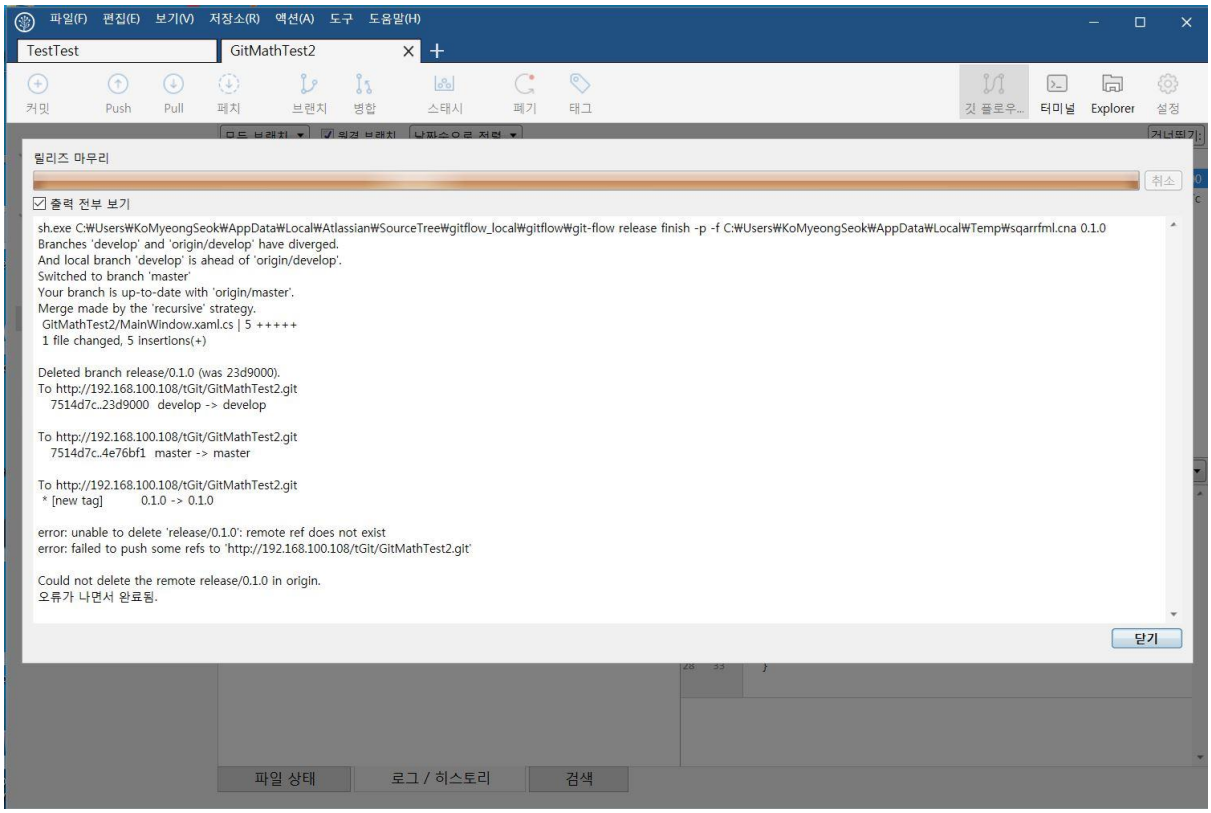


User1은 Release/0.1.0을 추가 해주도록 한다.



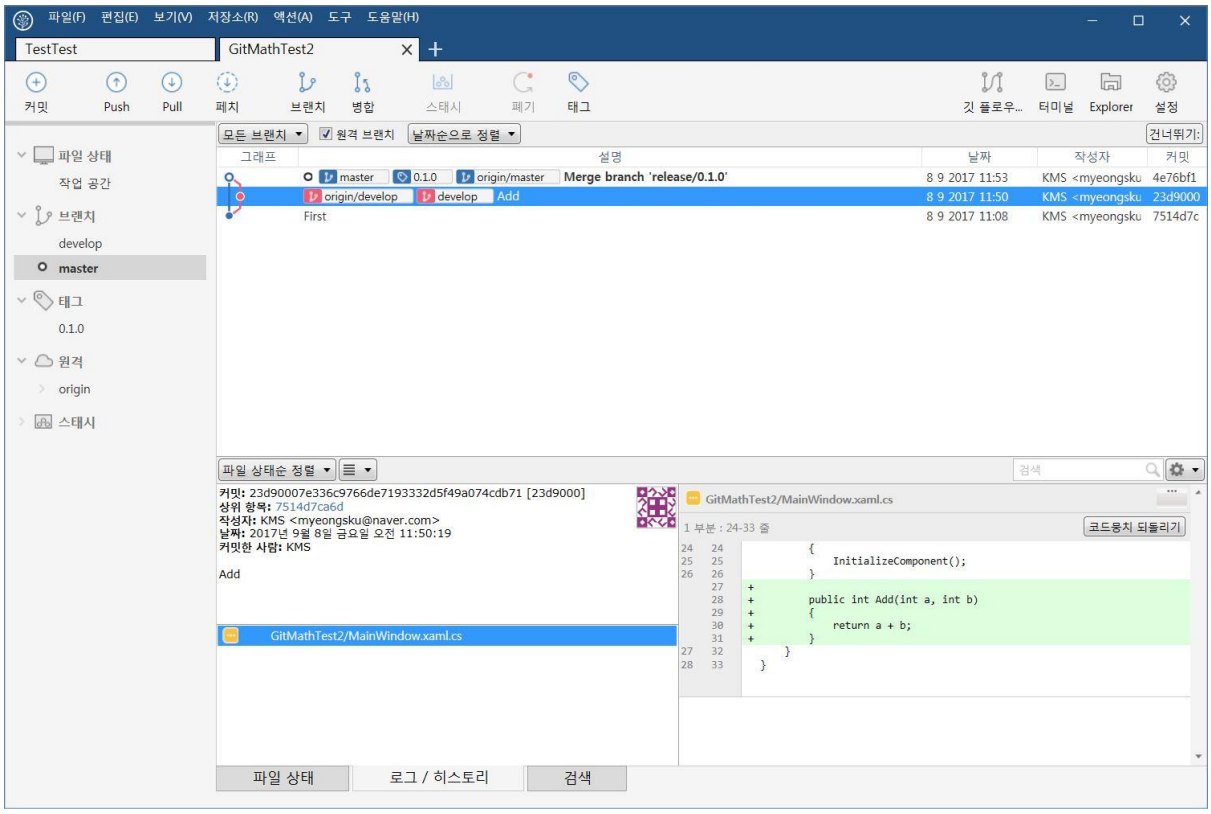
그 후 해당 Release가 개발이 끝났다면 릴리즈를 마무리 해준다.





위에 에러가 뜨면서 완료가 됐는데, 해당 에러는 origin 즉, 원격 저장소에 release/0.1.0 이라는 브랜치가 존재하지 않아서 해당 브랜치를 삭제할 수 없다는 얘기이다. 왜냐하면,

현 작업들은 굉장히 간단한 작업들이기 때문에 release를 start 후 따로 원격 저장소에 push를 해주지 않았기 때문이다. 고로, 해당 오류가 난다면 무시해도 된다.

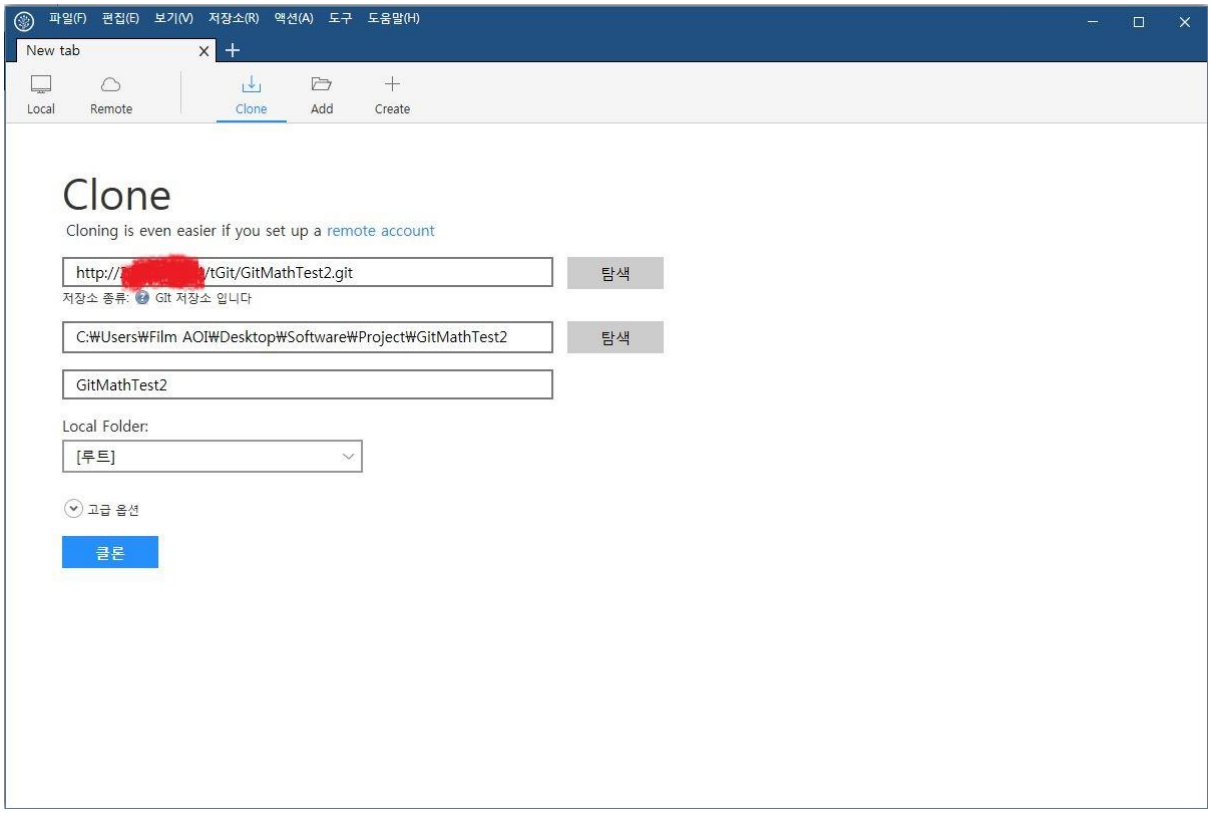


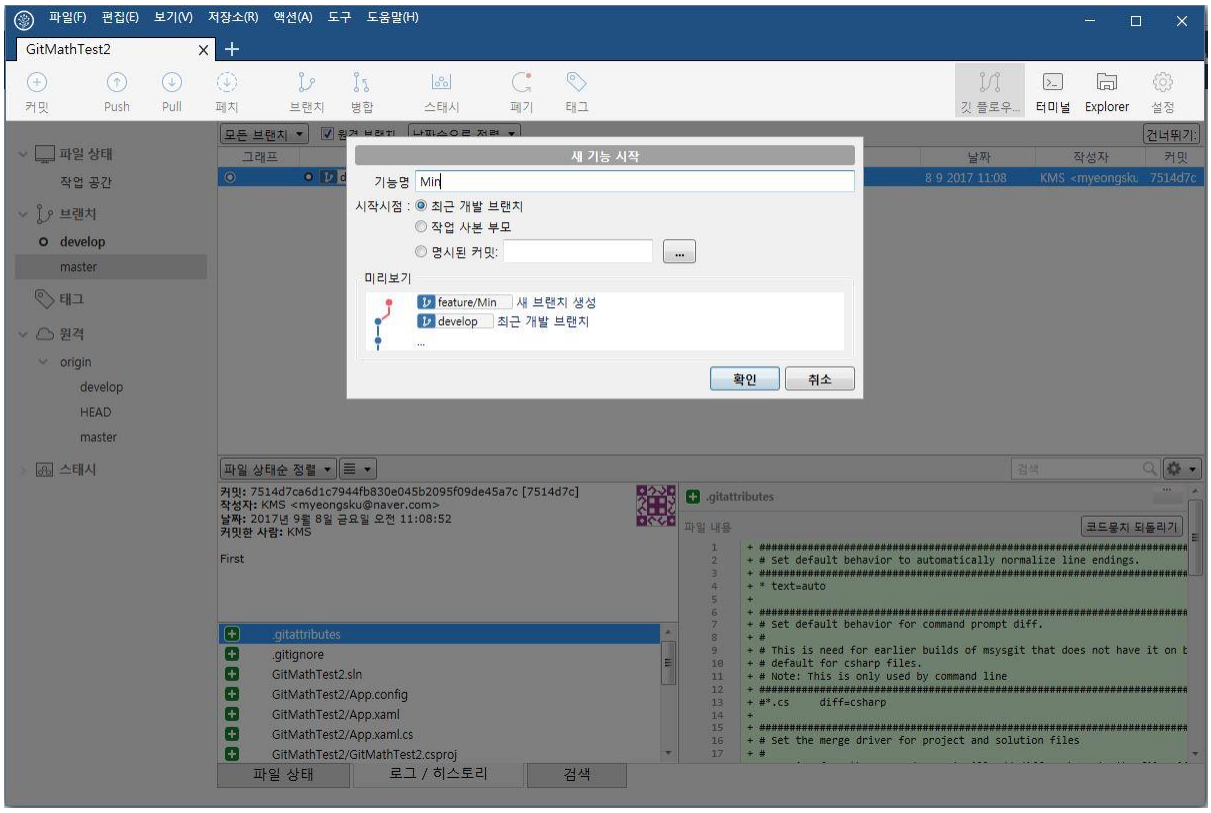
Release/0.1.0의 작업이 전부 끝났다.

사실, git은 Push와 Pull을 해주며 작업을 하게 되면 버전 관리가 굉장히 유연하다.

때문에 팀으로 프로젝트를 진행을 할 때 각자 할 작업을 하며 남이 작업 했던 부분을 멋대로 바꾸거나 하지만 않으면 꼬일 확률이 적다.

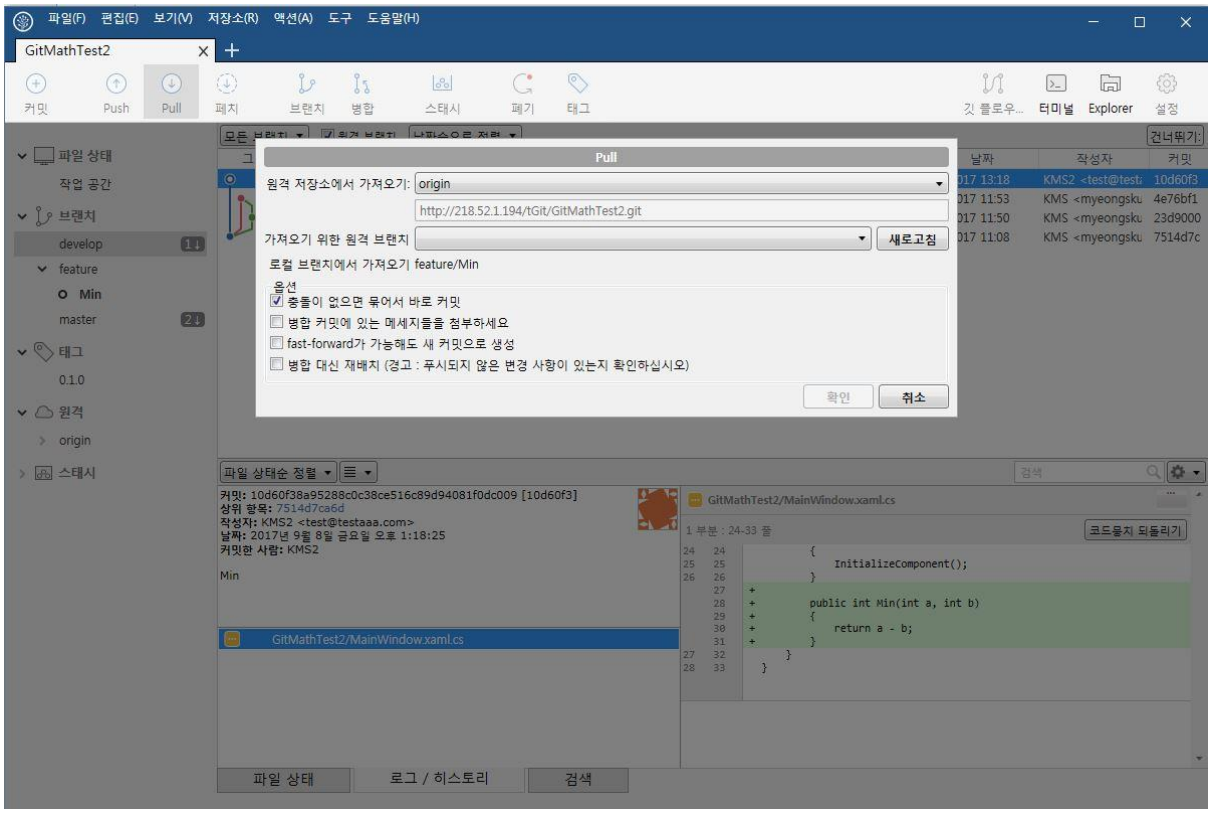
그럼에도 불구하고 소스코드가 꼬일 때가 있는데 그럴 때는 꼬인 부분을 git에서 찾아주기 때문에 직접 수정후 다시 커밋, 푸시를 해주게 되면된다. 이에 대한 것은 뒤에 추가 설명을 하도록 하겠다.





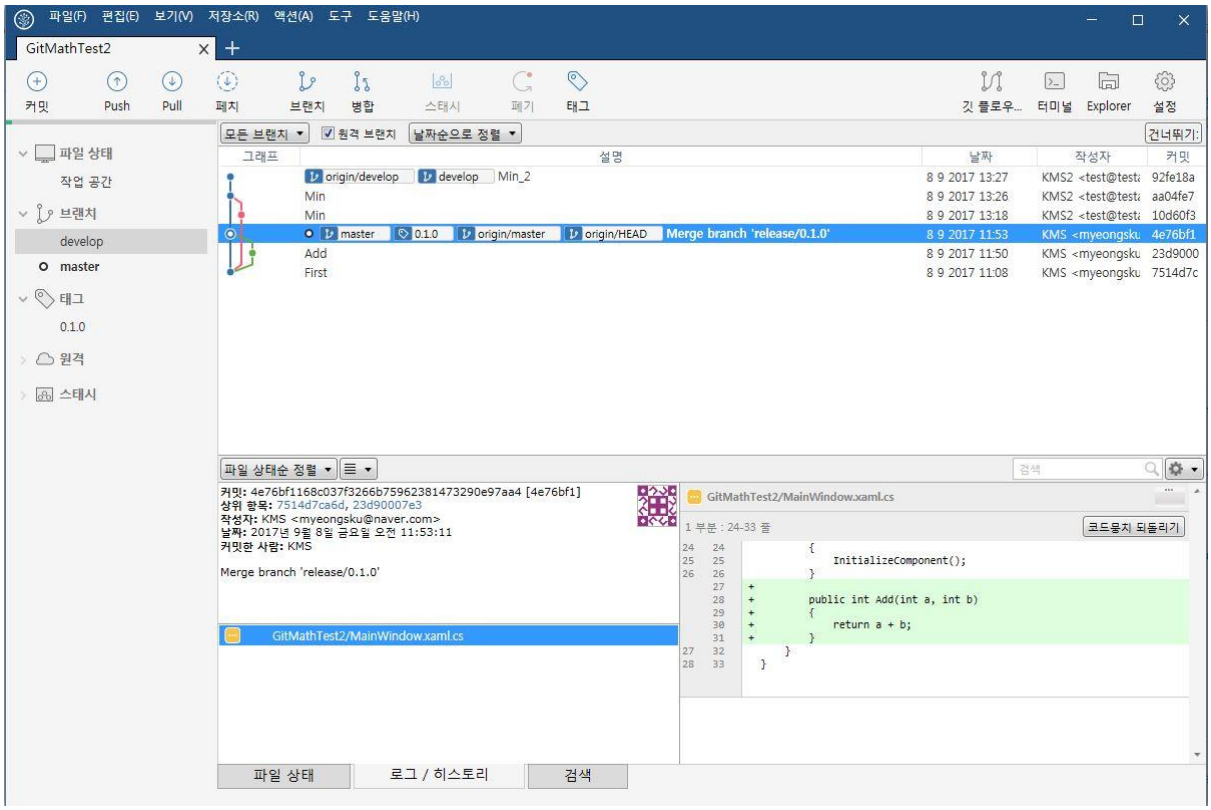
Clone을 하게 되면 git flow init을 다시 해줘야한다.

저장소로부터 Clone 후 Feature/Min을 User2로 추가를 해줬다.

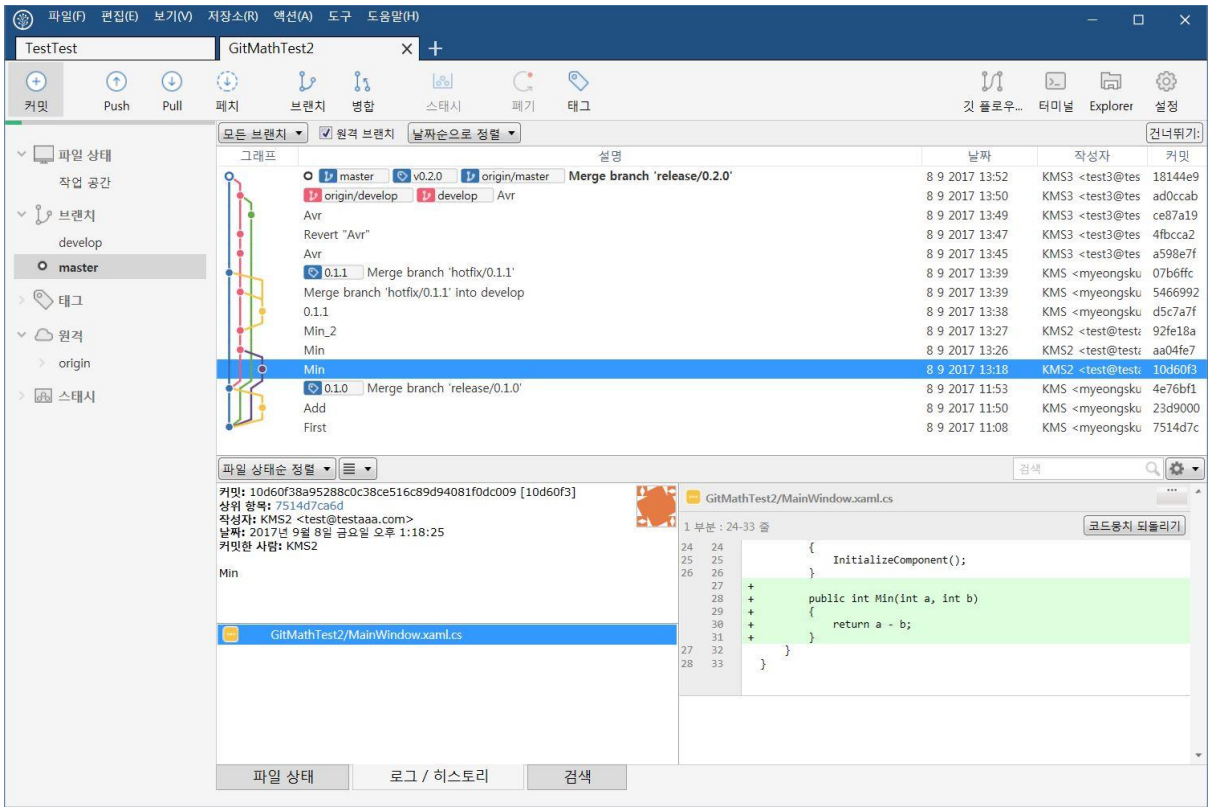


SourceTree에서 원격 저장소의 변화가 생기면 pull을 해야할지 위와 같이 나타낸다.

그러면 위의 사진처럼 해당 브랜치를 pull로 로컬 저장소에 merge 시켜주면 된다.



로컬 저장소에서 pull이 끝난 후 Feature/Add와 같이 Finish를 해줬다.



그 외 Avr 와 Hotfix 역시 별반 다를 것 없이 앞에 내용들과 같은 방법으로 진행을 하면 된다.

#Merge 작업(pull) 중에 충돌!

```
<<<<<<< HEAD
충돌 내용
>>>>>>> branchName
```

충돌이 발생을 하게되면 충돌이 발생한 소스 코드에 다음과 같은 표시와 함께 충돌된 코드가 보이게 된다. 이는 각 각의 브랜치에서 변경 내용이 같은 행이기 때문에 발생한 것이라고 볼 수 있습니다.

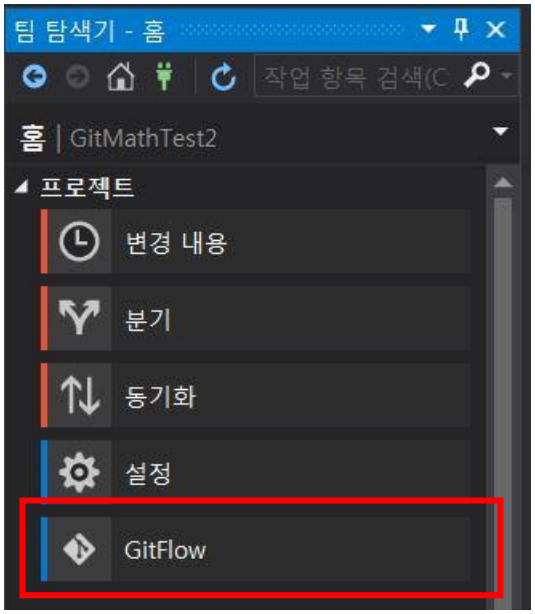
해당 내용을 직접 수정한 후 커밋 & 푸시를 해주게 되면 해결이 됩니다.

⊙How to use VisualStudio

Visual Studio 내에서도 플러그인으로 GitFlow기능을 확장할 수 있다.

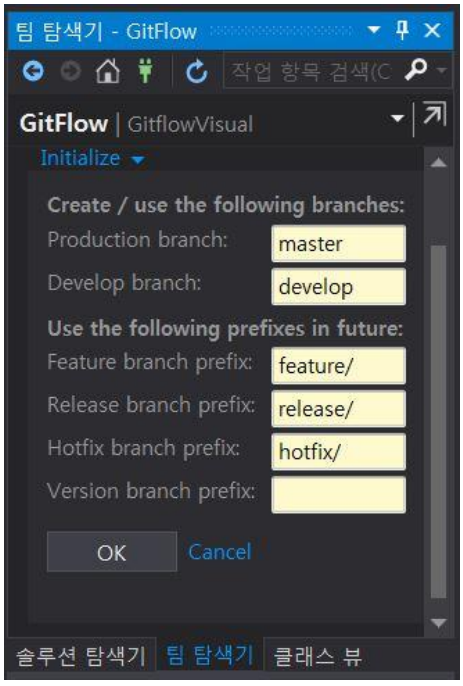
PluginSite

위의 하이퍼링크 사이트를 통해 다운 받은 후 Visual Studio를 다시 실행을 시키게 되면



=> 다음과 같이 'GitFlow'가 추가가 된다.

GitFlow를 클릭을 하게 되면,

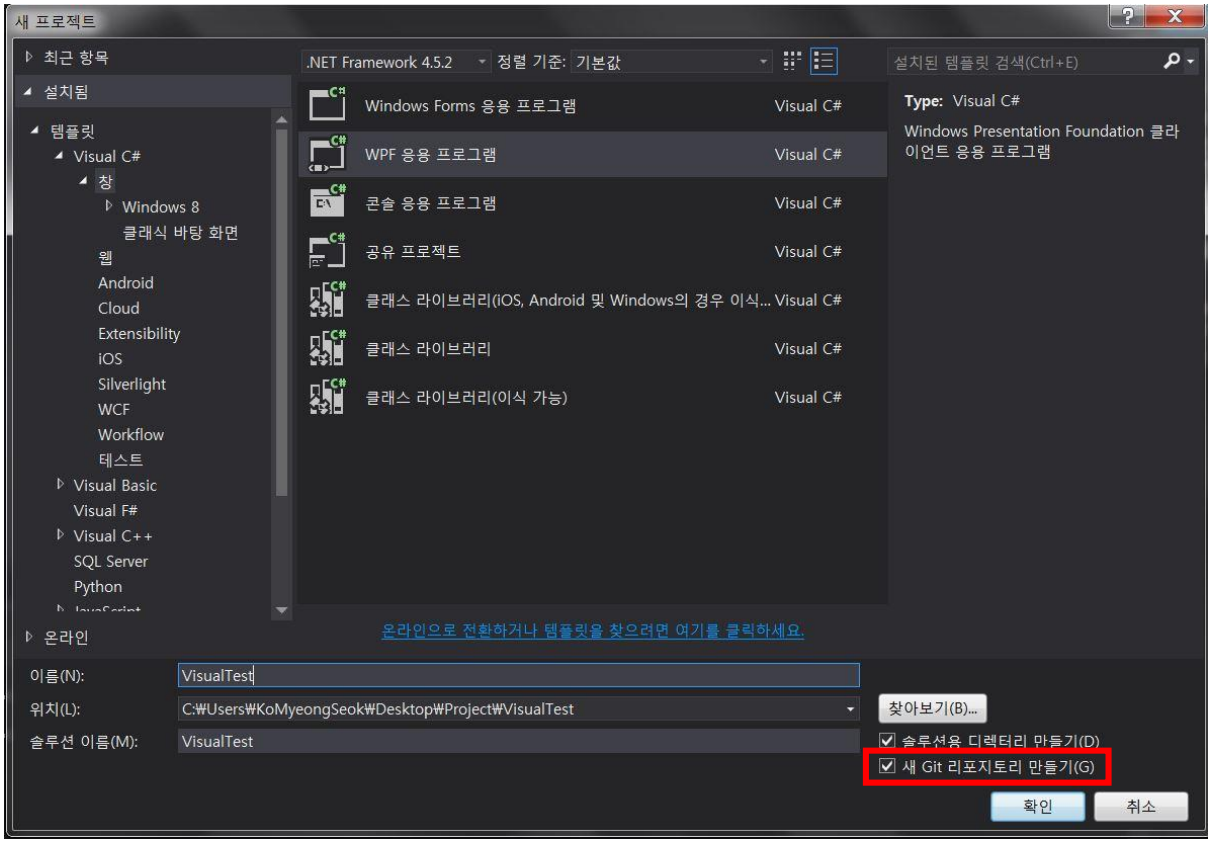


=> SourceTree와 같이 Initialize 창이 뜬다.

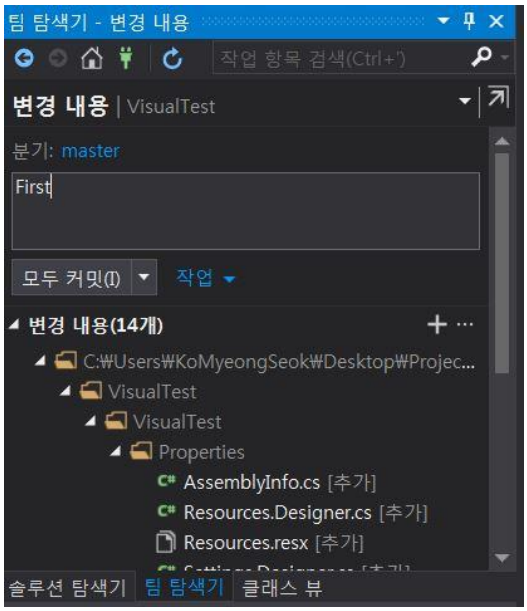
OK를 눌러 확인을 해주면 GitFlow 설정이 완료된다.

(단, 다른 과정들과 마찬가지로 git 이 먼저 Initialize 되어 있어야한다.)

만약, git이 Init이 되어 있지 않는다면,

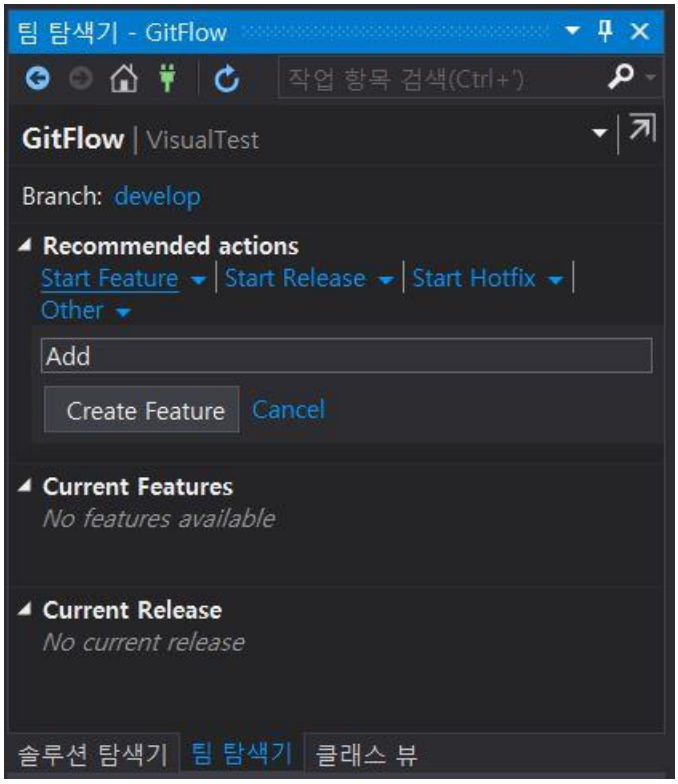


다음을 체크해 줘서 프로젝트 생성과 동시에 git을 init 시킬 수도 있다.



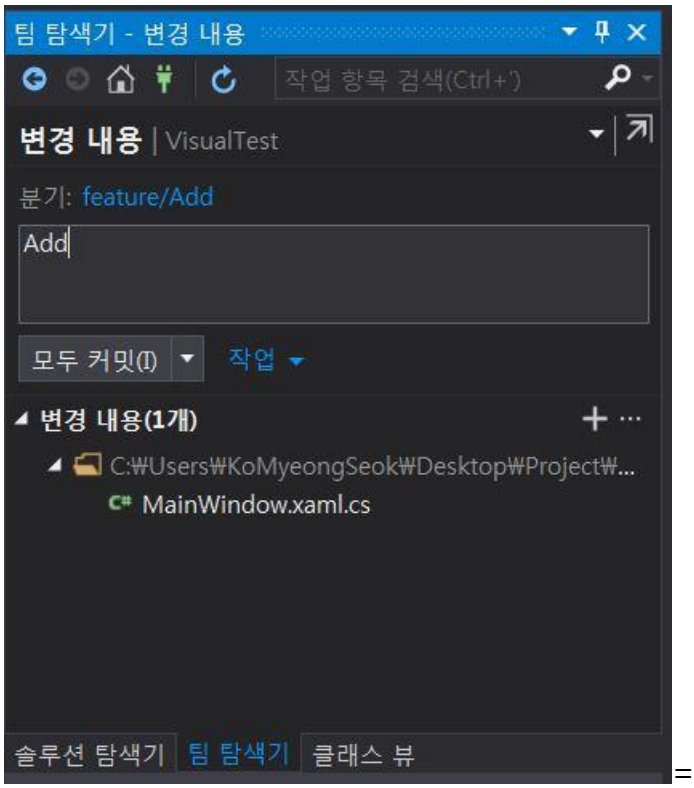
=> 프로젝트를 처음 만들어 준거라면 다음과 같이 커밋을 한 후 먼저 원격 저장소에 푸시 해준다.

푸시와 git flow init 과정까지 모두 끝냈다면 Feature 브랜치를 하나 추가해보도록 한다.

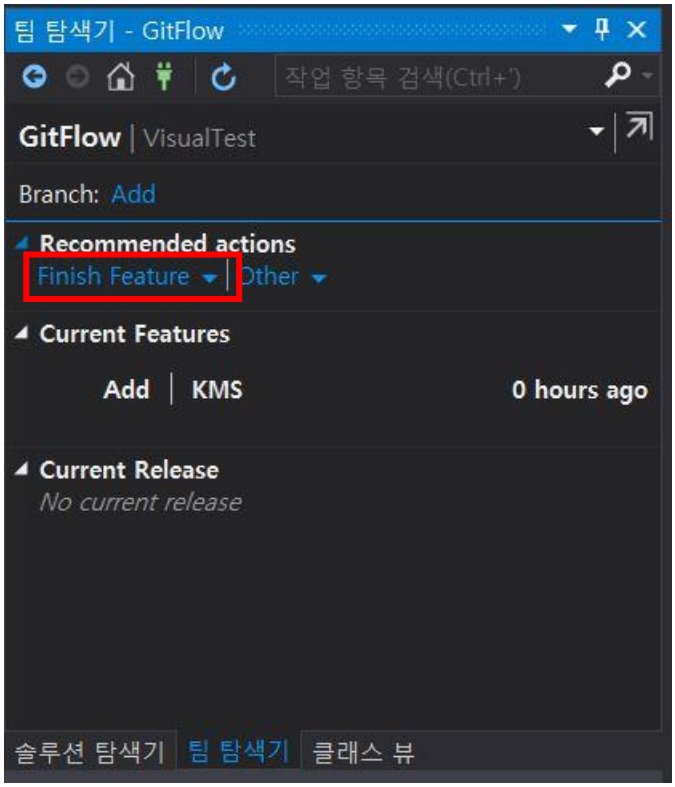


'Start Feature'를 클릭하고, 기능의 이름을 지정해준 후 'Create Feature'을 통해 'Feature/BranchName' 브랜치를 만들어준다.

그 후, 해당 기능의 개발이 끝나면

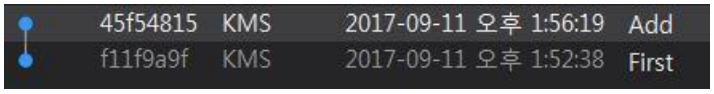


=>다음과 같이 커밋을 해준후,



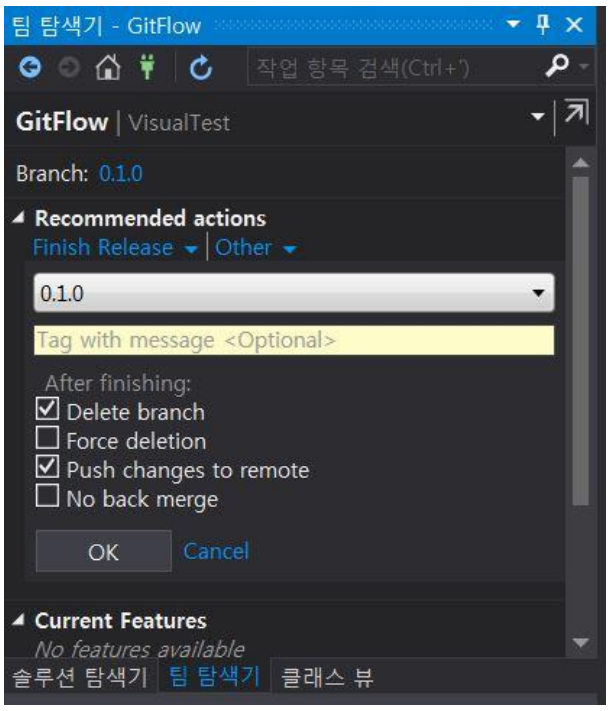
=>'Finish Feature'을 눌러준다.

그 후, OK를 눌러 완료를 해준다.



완료를 한 후 기록을 확인을 하면 다음과 같이 나온다.

Feature과 같은 방법으로 Release의 브랜치를 추가 해주고(이름은 현재 개발에 맞는 버전 넘버링을 추천.) Finish Release 를 해준다.



=> Finish Release를 해줄 때, Tag명은 꼭 해당 버전 넘버링으로 넣어주고, 옵션에서 'Push changes to remote'를 체크해 원격 저장소의 'master'와 'develop'에 푸시를 해준다.

Finish Release를 한 후 원격 저장소에 푸시를 하고 난 기록이다.

위와 같이 Visual Studio 자체에서 GitFlow 플러그인을 이용하여 제어가 가능하지만, 새로운 브랜치를 추가를 해준 후 커밋을 실수로 안하고 Finish를 하게 되버리면 작업이 꼬여버리는 단점이 있다. Merge 작업시 충돌이 났을 때 수정 작업은 굉장히 편하다.

SourceTree는 자체적은 조작이 편하고, 위와 같은 실수를 하게 되면 에러를 통해 사전에 방지를 해주지만, 충돌이 났을 때의 수정이 불편한 점이 있다.