공학 트렌드

2014. 3. 00. [제00호]

GIT Flow를 활용한 효과적인 소스 형상 관리 Part 3 : Source Tree를 이용한 GIT Flow 실습

소프트웨어공학센터 경영지원TF팀

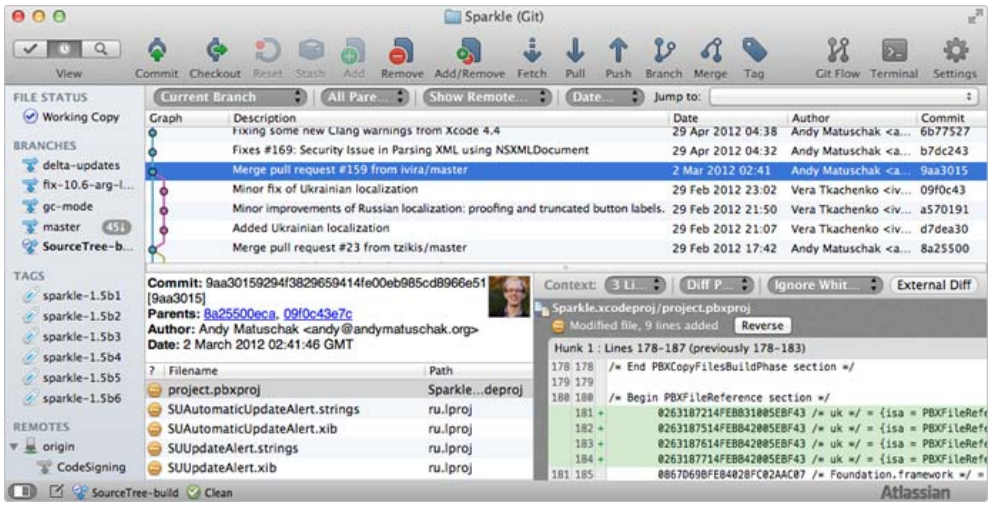
Contents

정보통신산업진흥원 부설 nipa 소프트웨어공학센터

그동안 터미널(Terminal)에서 GIT Flow를 사용하는 법을 학습했다. GIT Branch 전략을 매번 터미널에서 작업하기란 쉽지 않다. 그러나 UI 툴을 이용한다면 빠르게 GIT 명령어와 GIT Flow 명령어를 통해 능률을 높일 수 있다. 이번 원고에서는 대표적인 GIT UI 툴인 Source Tree(http://www.sourcetreeapp.com)를 이용해서 쉽게 GIT 명령어뿐 아니라 GIT Branch 전략을 사용해 효율을 높이도록 실습하고자 한다. 본고에서는 소스 간의 충돌을 최소화하여 효율적인 개발이 가능해 특히 대규모 인원의 개발에서 주로 쓰이는 GIT Flow를 소개하고, 예제를 통해 실무 활용 방안을 살펴보고자 한다.

Source Tree는 〈그림 1〉과 같이 좌측 화면에서 Branch, Tag 정보를 가운데 화면에서는 히스토리(History) 이력, 하단에서는 무엇을 수정했는지 파일 이력, 우측 하단에서는 파일 diff 이력을 보며 개발의 편이성을 높인 툴이다. 자주 쓰는 GIT 명령어는 상단 아이콘 바(Bar)에 위치하고 있어서 터미널 없이도 개발할 수 있는 UX를 제공하고 있다.

그림 1_Source Tree UI 화면

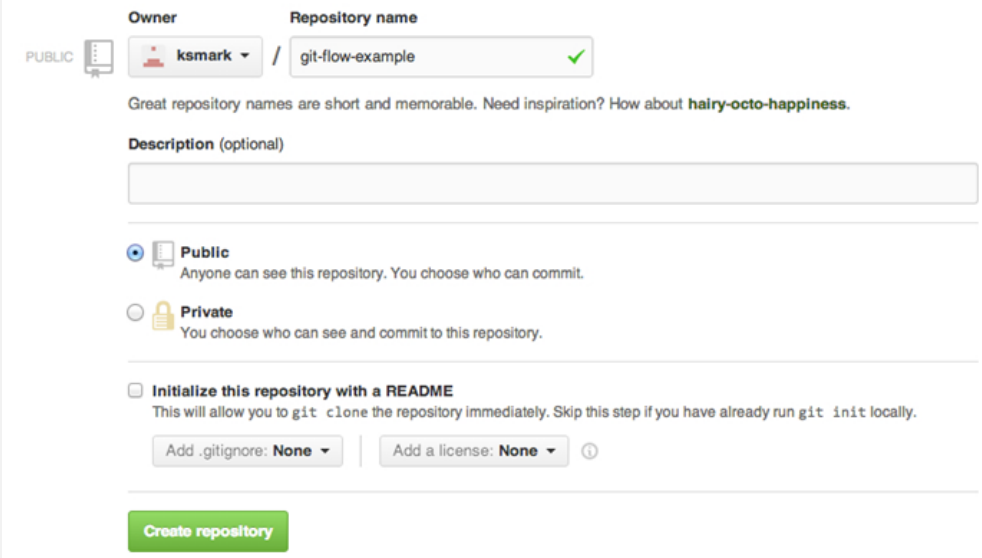


Ⅰ. 설치

1. GIT Hub Project 생성

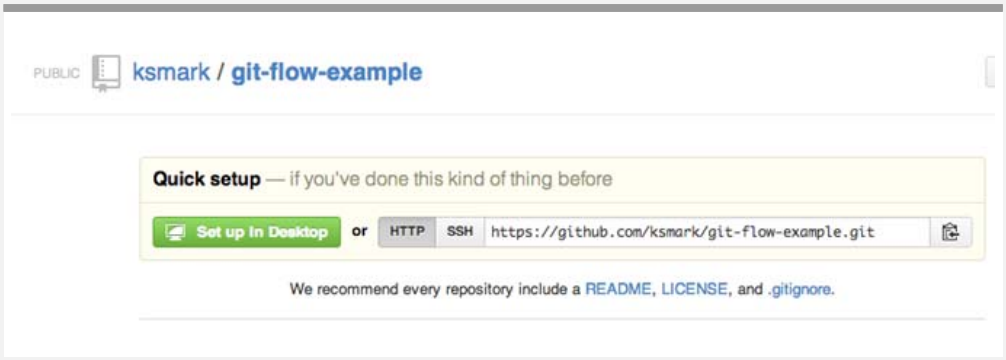
〈그림 2〉와 같이 GIT Hub Project(http://github.com)에 접속하여 GIT Repository를 생성한다.

그림 2_GIT Hub Repository 생성 UI



Repository를 생성하면 〈그림 3〉의 화면과 같이 GIT Repository 주소를 알 수 있다. 나중에 Source Tree 환경설정 화면에서 이 주소를 입력하여 Clone 시키면 된다.

그림 3_GIT Hub Repository 생성완료 UI



다음은 인증 절차이다. GIT Hub는 SSH 인증방식을 사용하고 있다. 처음 GIT Hub 사용자라면 인증하는 부분에서 시간이 소요된다. 인증 절차를 잘 통과해야 GIT Repository나 Source Tree를 문제없이 쓸 수 있다. Local PC에서 GIT Repository 접근을 위해서는 SSH 인증 방식을 따라야 한다. GIT Hub Repository에 인증/접속할 수 있는 SSH 키(key) 발급은 다음의 URL을 참조하여 완료하도록 한다.

https://help.github.com/articles/generating-ssh-keys

http://git-scm.com/book/en/Git-on-the-Server-Generating-Your-SSH-Public-Key

터미널에서 작업 디렉토리를 생성하고 GIT Repository 초기화를 진행한다.

```
$ cd /development/work/
$ mkdir /development/work/git-flow-example
$ cd git-flow-example
$ touch README.md
$ git init
$ git add README.md
$ git commit -m "first commit"
$ git remote add origin https://github.com/ksmark/git-flow-example.git
$ git push -u origin master
```

2. Source Tree 설치

Source Tree는 개인이나 기업이 쉽게 쓸 수 있는 툴이다. 오픈소스 라이센스는 아니지만 EULA 동의 하에 무료로 사용이 가능하다.(https://answers.atlassian.com/questions/55442/sourcetree-license) 이 툴은 Window OS와 MAC OS에서 사용 가능한 툴이며, Linux OS는 아직 지원하지 않고 있다.

Source Tree(http://www.sourcetreeapp.com) 웹 사이트에 접속해서 Source Tree 무료 버전을 다운로드 한다. 참고로 MAC OS 사용자라면 주의해야 할 사항이 있다. 〈그림 4〉와 같이 현재 Mac OS의 앱 스토어(App Store)에 있는 Source Tree의 버전은 1.5.6 이다.

그림 4_App Store의 Source Tree 버전

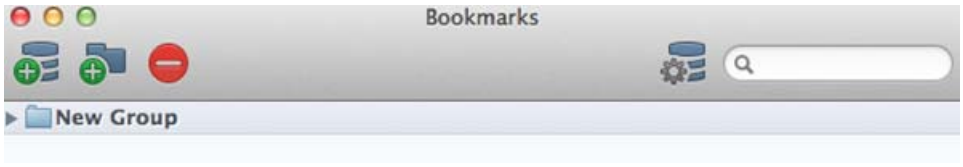
그러나 여러 가지 이슈(http://blog.sourcetreeapp.com/2012/02/16/abandoning-the-mac-app-store/)들로 인해서 Source Tree의 최신 버전은 Mac OS의 App Store에서 다운로드 할 수 없다. 다음 〈그림 5〉의 Source Tree(http://www.sourcetreeapp.com) 웹 사이트에 직접 접속해서 사용하면 가능하다. 2014년 2월 현재 Source Tree의 최신 버전은 1.8.1이다.

그림 5_Source Tree 다운로드 하는 웹 화면

3. Sourc Tree 환경 설정

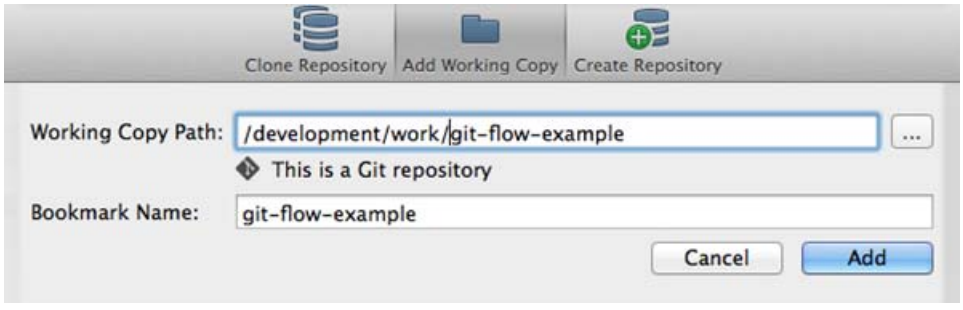
필자는 Mac OS 기반으로 설명하고자 한다. Source Tree를 실행하면 〈그림 6〉의 Bookmarks 화면을 볼 수 있다. 아이콘 바의 폴더 아이콘을 선택하고 폴더 이름을 본인의 스타일에 맞게 저장한다.

그림 6_Bookmarks 화면



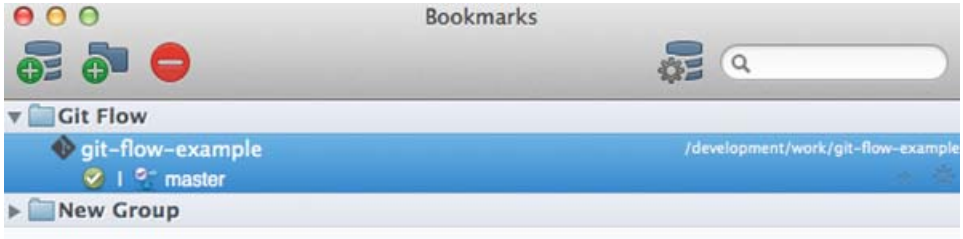
그리고 두 번째 폴더 아이콘(Add Working Copy)을 선택하고 〈그림 7〉과 같이 2.1에서 생성 및 clone 했던 GIT Hub Repository로부터 Clone 시켰던 Path를 입력한다. 이 화면에서 GIT Repository로부터 Clone 할 수 있고, GIT Repository를 생성할 수 있다. Local Path를 명확하게 잘 지정하면 자동으로 'This is a Git repository' 라는 문구를 확인 할 수 있다.

그림 7_Bookmark의 Repository 생성화면



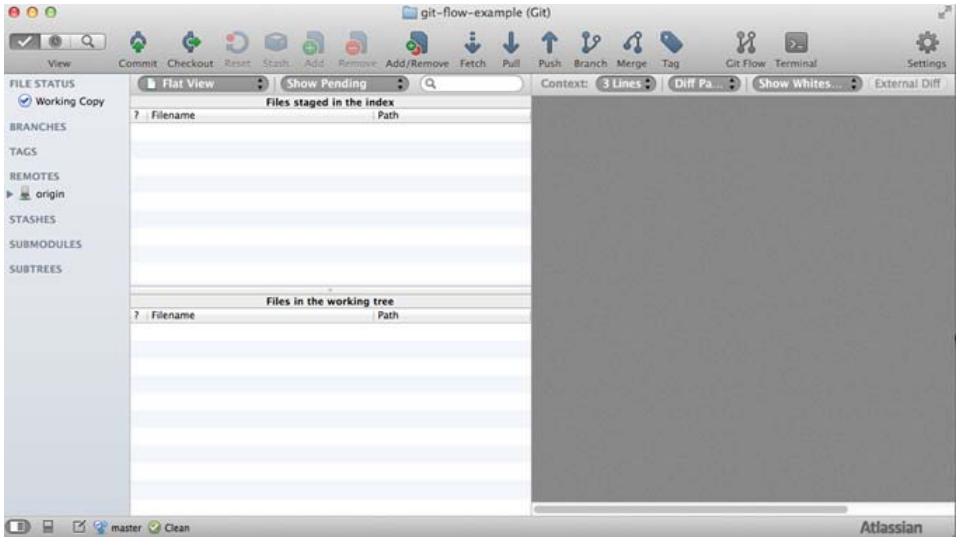
정상적으로 작동이 되면, 〈그림 8〉의 화면을 볼 수 있다.

그림 8_입력한 Bookmarks 화면



입력한 git-flow-example Bookmark를 Open하면 〈그림 9〉의 Source Tree 기본 화면이 나온다.

그림 9_GIT Repository 첫 화면



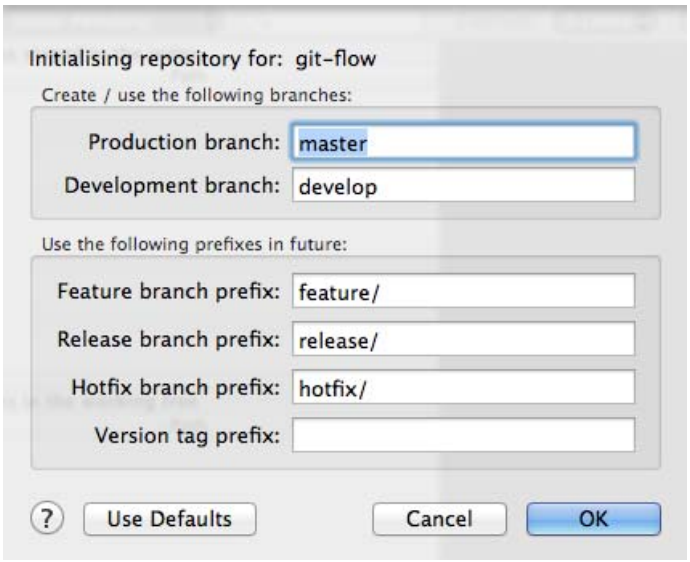
4. 초기화

그림 10_아이콘 바에 있는 GIT Flow 아이콘

GIT Flow 전략을 사용하려면 두 가지가 있다.

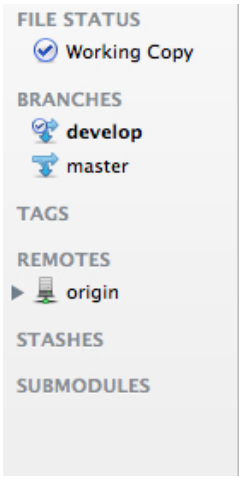
상단 우측에 있는 〈그림 10〉과 같이 Git Flow 아이콘을 클릭하거나, 메뉴의 Repository 〉 Git Flow / Hg Flow 〉 Initialise Repository를 선택한다.

그림 11_GIT Flow Repository 초기화 화면



〈그림 11〉과 같이 Production branch(master), Development branch(develop)와 Branch Prefix를 수정할 수 있는 팝업 창이 보이면, 확인하고 OK 버튼을 누른다. Source Tree는 GIT Flow를 쓸 수 있는 Repository 초기화를 진행한다. 참고로 대부분 회사에서는 기본 디폴트를 주로 사용하고 있으나, 특정 회사에서는 눈에 잘 보이기 위해서 Branch Prefix를 '/'(디렉토리) 대신 '-'으로 사용하고 있다. (예. 'feature/' 대신 'feature-')

그림 12_GIT Flow 초기화 화면의 좌측



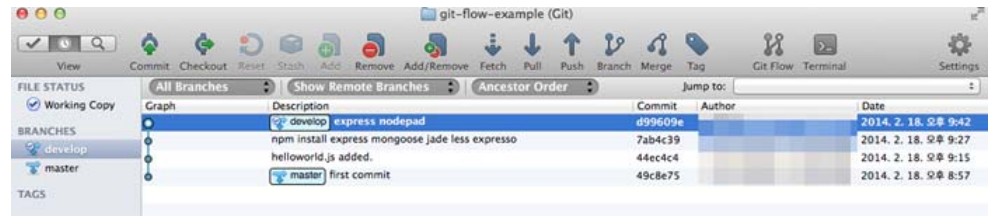
GIT Flow 로 초기화되면, Develop Branch가 새로 생성되고 개발 위치는 Develop Branch로 이동한다. 〈그림 12〉와 같이 좌측 화면처럼 BRANCHES의 develop로 볼드체, 아이콘 마크가 구별되어 표현하고 있다.

II. 예제를 이용한 Source Tree / GIT Flow 적응

1. Develop Branch

사소한 작업은 Develop Branch에 작업을 진행하고 Add/Remove 아이콘을 선택하고 Commit 아이콘을 선택하여 작업을 진행한다. Graph를 보면 하나의 작업을 진행하고 있음을 보여주고 있다.

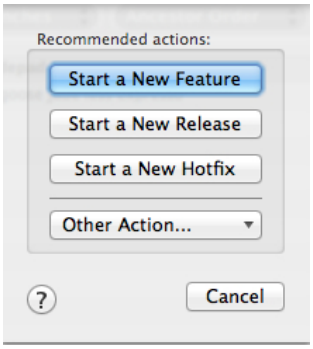
그림 13_Develop Branch 작업



2. Feature 생성

이제 새로운 Feature를 생성한다. GIT Flow 아이콘을 클릭하면 다음의 팝업을 볼 수 있다. 새로운 Featue, Release, Hotfix를 시작할 수 있도록 한다. 'Start a New Feature' Action을 선택한다.

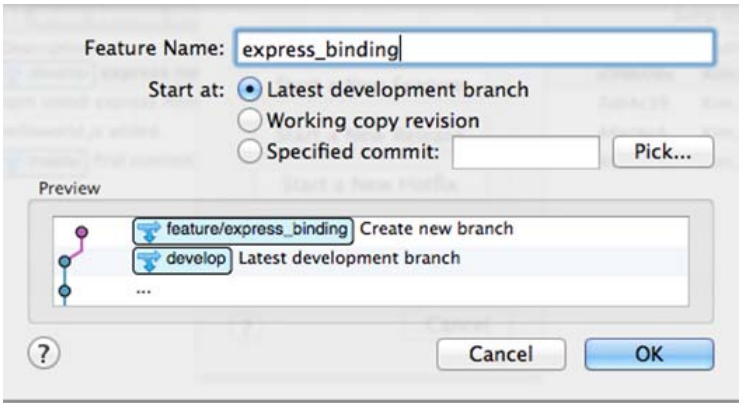
그림 14_GIT Flow 아이콘 실행 화면



Feature 이름을 넣는 팝업창 하나가 뜨면, 사용할 Feature 이름을 쓴다. 〈그림 15〉와 같이 'feature/express_binding' 이라는 Feature를 생성한다. 생성되는 기준은 Develop

Branch를 바탕으로 한다. 팝업의 내용을 보면 Develop Branch의 특정 Commit 버전을 바탕으로 Feature 생성이 가능함을 보여주고 있다.

그림 15_Feature 생성 화면



Feature를 생성하면 〈그림 16〉과 같이 Feature 디렉토리와 그 밑에 새로운 'express_binding'이라는 Feature가 생성된 것을 볼 수 있다. 기준 작업 Branch는 'express_binding'이 된다.

그림 16_Feature 생성 후 좌측 화면

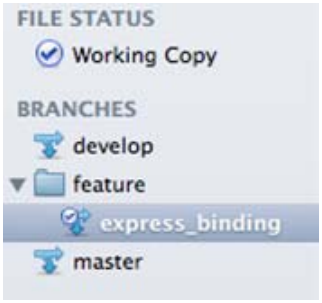
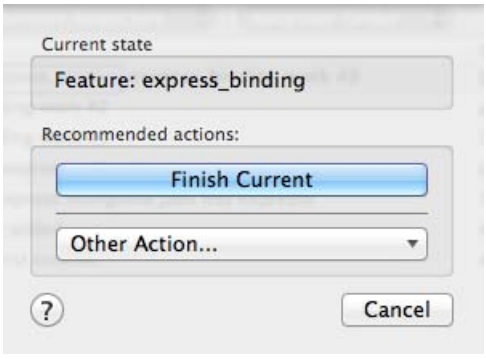


그림 17_Feature 종료 화면

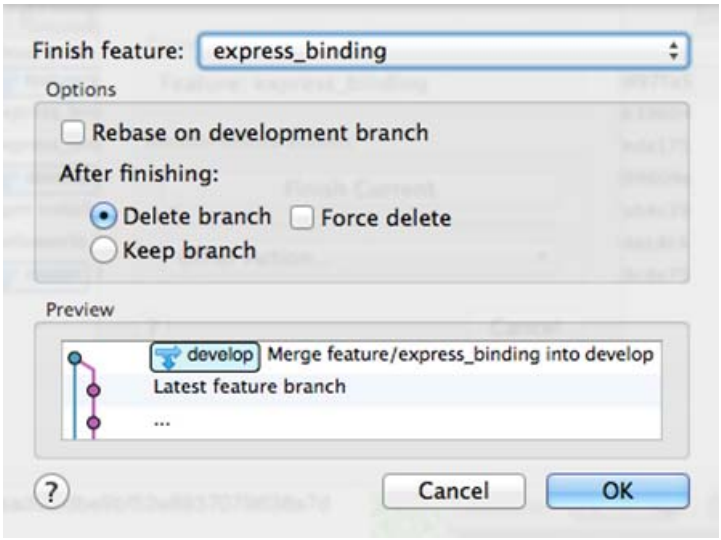


새로 만들어진 feature 의 여러 번의 Commit 작업 후 Develop Branch에 Merge을 실시

한다. GIT Flow 아이콘을 선택하면 〈그림 17〉과 같은 화면이 나온다. Finish Current 액션을 선택한다.

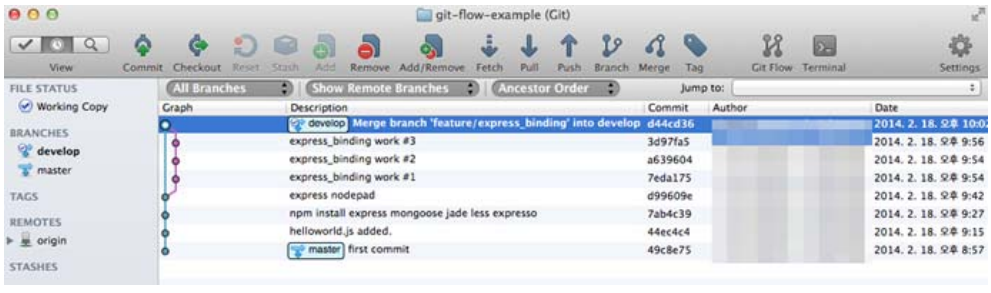
'feature/express_binding' Feature Branch는 Develop Branch에 Merge 된다. 이 때, Source Tree는 〈그림 18〉과 같이 다양한 옵션을 제공한다. GIT Merge 명령어 대신 GIT Rebase 명령어로 사용할 수 있고, Feature를 지우거나(또는 강제로 지우거나) Feature Branch를 지우지 않고 계속 유지시킬 수 있는 옵션을 제공한다. 사용하고 있던 express_binding Feature Branch를 삭제하는 것으로 가정하고 진행한다.

그림 18_Feature Branch 종료 화면



〈그림 19〉 를 보면, Feature Branch는 삭제되었고, GIT History 내역까지 잘 적용되었다. History 상 Feature Branch 에서 작업했던 내용이 분홍색이고, 기존에 작업했던 Develop Branch는 파란색이다. Merge 되면서 하나로 묶여진 것을 확인할 수 있다.

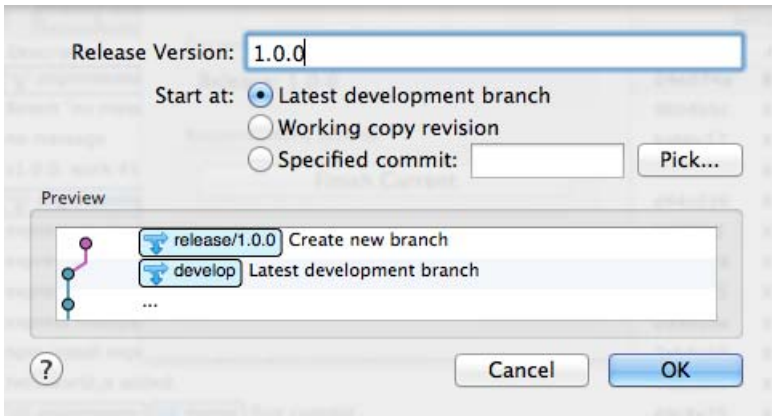
그림 19_Merge 이후 화면



3. Release Branch

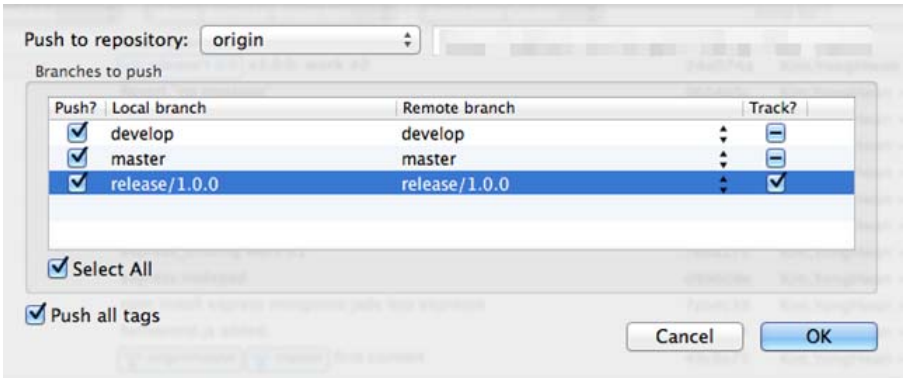
GIT Flow 아이콘을 선택한 후, "Start New Release"를 선택한다. 〈그림 20〉과 같이 release version을 입력한 후 OK 버튼을 눌러 생성한다. Feature Branch와 마찬가지로 Develop Branch를 기반으로 작업할 수 있고, 마찬가지로 Develop Branch의 특정 commit 버전을 바탕으로 Release Branch를 생성할 수 있다. 버전은 1.0.0으로 지정한다.

그림 20_Release Branch 생성 화면



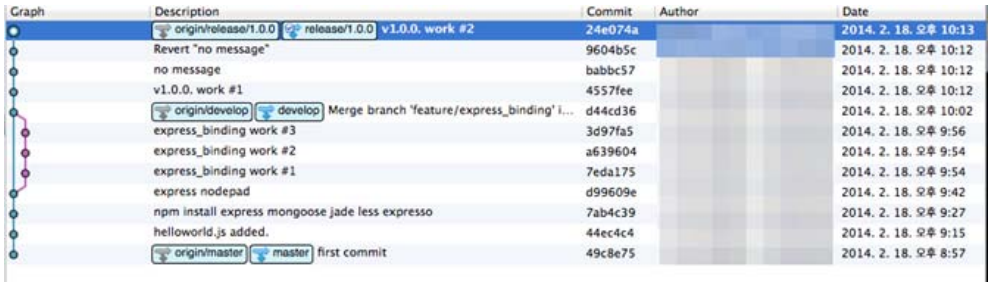
새로운 'release/1.0.0' branch가 생겼다. 여러 commit 작업을 완료한 후, 지금까지의 모든 작업을 GIT Server로 Push한다. Push 할 때, 아이콘 바에 있는 'push' 아이콘을 선택한다. 〈그림 21〉과 같은 팝업창이 뜨고 어느 Local Branch를 Push 할 것인지 결정할 수 있다. 'push' 버튼을 눌러 push 하지 않은 Develop, Master, release/1.0.0 Branch를 모두 push한다.

그림 21_Push 팝업 화면



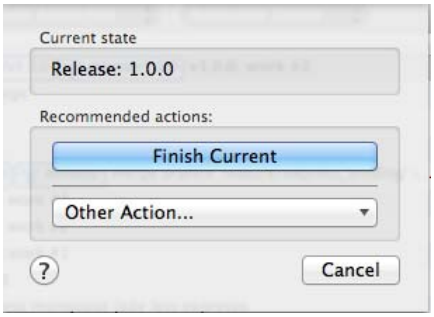
Push를 진행하면 〈그림 22〉와 같이 'origin/release/1.0.0', 'origin/develop', 'origin/master'와 같이 3개의 Remote Branch가 생겼고, Local Branch도 같이 맞춰졌다.

그림 22_Push 이후 화면



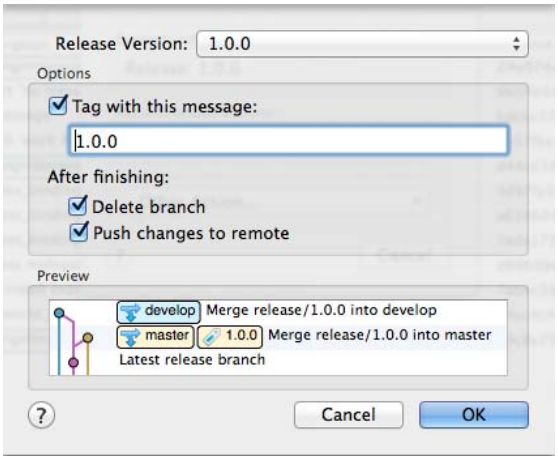
이제 release 1.0.0 개발이 다 완료되었고, 개발을 종료할 예정이다. 'Git Flow' 아이콘 버튼을 누르면 〈그림 23〉과 같이 팝업창이 뜬다. 'Finish Current'를 선택한다.

그림 23_Release 종료 화면



터미널에서는 Release 를 종료할 때 release/1.0.0 Branch는 삭제되고 Master와 Develop Branch에 Merge가 된다. 그러나 Source Tree에서는 좀 더 기능이 있는 형태를 제공하고 있는데, 〈그림 24〉와 같이 tag에 대한 옵션과 Merge 이후 Release Branch를 삭제할지, Remote 서버에 push 할지를 결정할 수 있다.

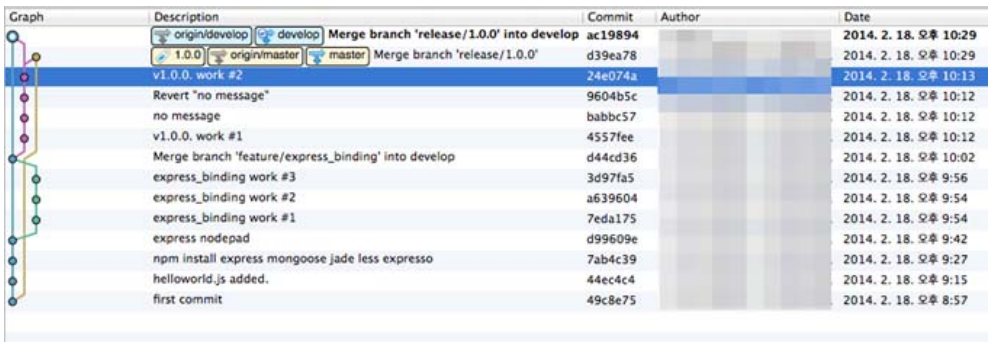
그림 24_Release 종료 후 tag, Merge 할 여부를 묻는 화면



필자는 Release Branch를 Delete 하고 변경 내용을 Remote 서버에 있는 release/1.0.0에 반영하도록 한다. 이때 push changes to remote를 선택하여 Local 작업을 모두 서버에 반영하도록 한다. 즉, Push까지 모두 포함한 작업을 포함하는 작업을 하도록 한다.

Release Branch를 Merge 하면 1.0.0 태그 생겼고, Master와 Develop가 모두 Merge되고 Remote Server에 모두 push까지 되었다. 참고로 색상은 큰 의미는 없다. 어떻게 Merge 되었고 commit 된 버전과 연관 관계를 알려주는 보여주는 Graph 이다.

그림 25_release/1.0.0 Merge 이후 화면

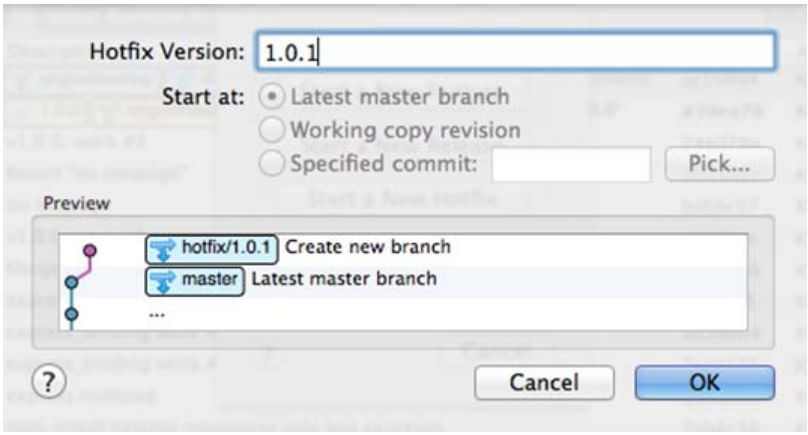


이제 Master 버전을 가지고 운영 서버(production)에 배포한다. 참고로 Develop Branch나 Release Branch는 개발 서버(develop)에 배포하고 테스트를 진행한다.

3. Hotfix

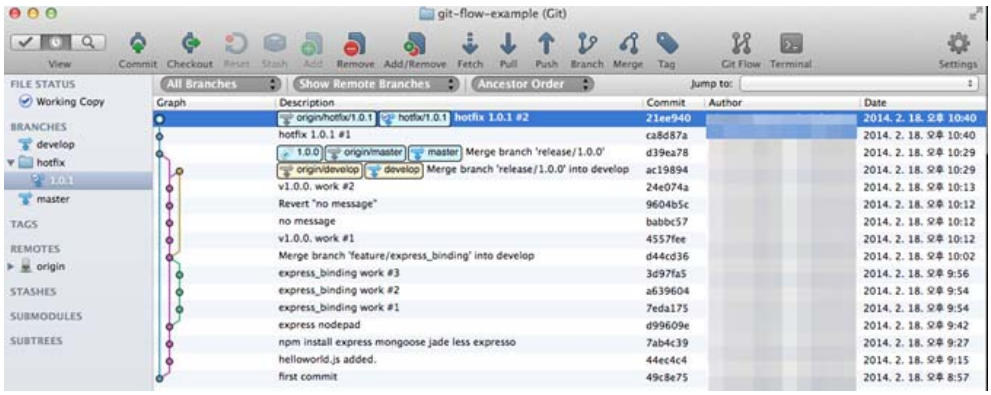
release/1.0.0에 버그가 생겨서 빠르게 패치(Patch)해야 할 일이 생겼다. GIT Flow 아이콘 버튼을 선택하고 〈그림 26〉과 같이 Hotfix 1.0.1을 만든다.

그림 26_Hotfix Branch 생성 화면



버그를 패치 할 소스를 수정 후 commit하고, Push 한다. Push하고 나면 〈그림 27〉처럼 Master 기반으로 수정된 버전임을 확인할 수 있다. 또한 Develop Branch는 Hotfix와 연관 관계가 없는 형태로 되어 있음을 확인할 수 있다.

그림 27_Hotfix push 후 화면



hotfix 1.0.1을 Develop Branch에 Merge하기 위해 GIT Flow 아이콘 버튼을 선택한다. 〈그림 28〉과 같이 팝업이 뜨면 'Finish Current'를 선택한다.

그림 28_Hotfix 종료 화면

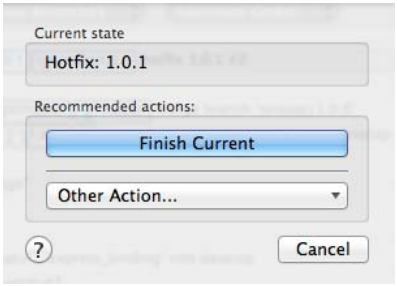
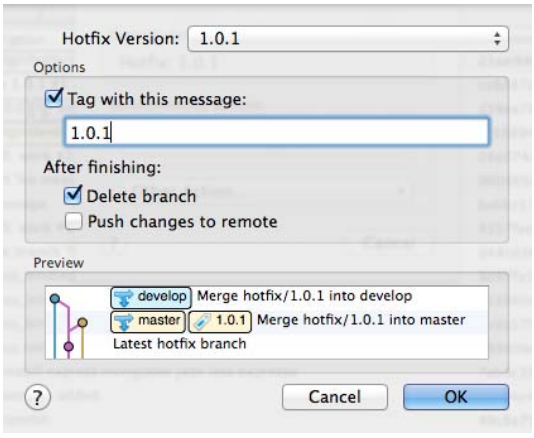


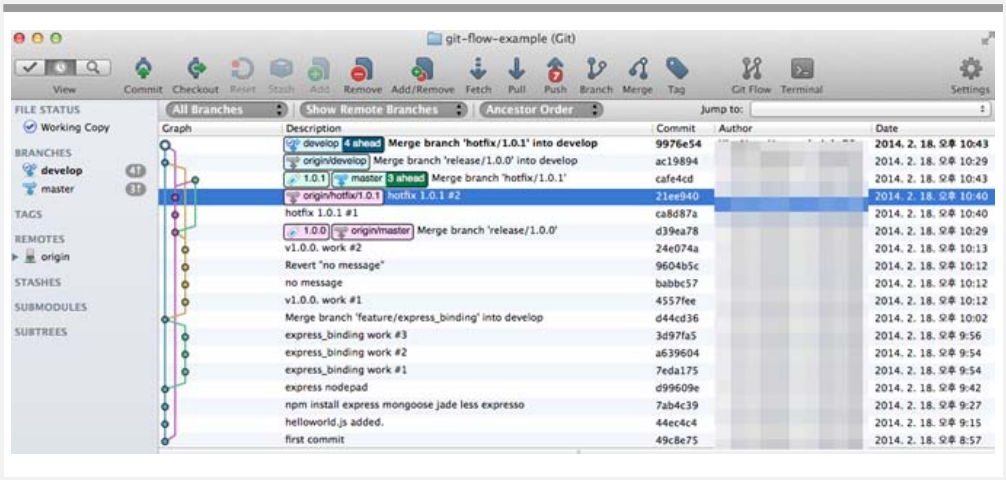
그림 29_Hotfix 종료 화면



그 다음 화면으로 〈그림 29〉의 tag와 Merge 이후 Hotfix Branch를 지울지, Remote 서버에 소스 commit 할지에 대한 팝업이 뜬다. 만약 'push changes to remote' 체크 박스를 체크하지 않으면 Local에서 Merge하고 push 하지 않는다.

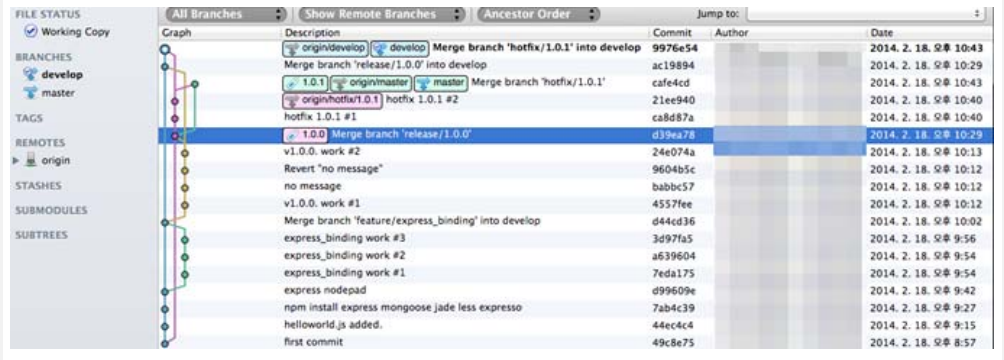
hotfix/1.0.1을 종료하면 Master와 Develop Branch에 Merge가 되고, 〈그림 30〉처럼 push 하지 않은 소스가 7개가 있다고 아이콘 버튼상에서 보여주고 있다.

그림 30_Hotfix Merge된 화면



Merge 된 Develop과 Master를 push한다. 〈그림 31〉과 같이 소스가 모두 GIT Server와 동기화 된 화면을 볼 수 있다.

그림 31_Remote 서버로 push 한 화면

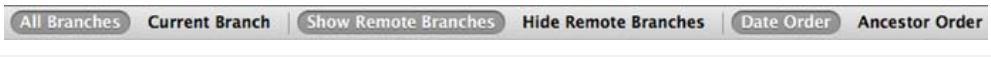


운영 서버(production)에 Master Branch를 가지고 배포를 진행하면 된다.

Ⅲ. 버전 단위로 보기

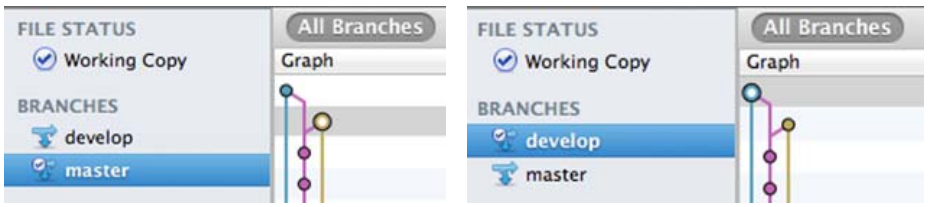
많은 개발자들이 하나의 프로젝트에서 Release, Hotfix, Feature, Develop, Master Branch가 존재하면 Source Tree 상의 Graph가 복잡해 보인다. 이럴 때는 Commit Description 화면 위에 〈그림 32〉와 같은 화면을 이용하면 된다.

그림 32_Branch 단위로 commit history 조정



만약 Current Branch를 선택하면 선택한 Branch의 가장 최신 버전에 큰 원이 생기고, 그 원을 따라 과거의 commit log history 상태를 보여준다. 〈그림 33〉은 Master를 선택했을 때, Develop Branch를 선택했을 때 각각의 Branch 최종 버전을 Graph에서 보여주는 것을 알린다.

그림 33_'Current Branch'를 선택했을 때 보여주는 Graph 상황



'Show Remote Branches' 또는 'Hide Remote Branches'는 Remote Branch를 History상에 보여줄 것인지의 여부를 결정한다. 'Date Order'와 'Ancestor Order'는 시간 순서로 보여 줄지, 조상(ancestor) 순서 단위로 보여줄 지를 결정한다. 각 Branch에서 일어나는 commit 시간이 Rebase, Merge를 통해 보여주는 Graph도 좋지만, 때로는 branch 관점에 본 조상 순서 단위를 보는 것도 편할 수 있다. 이런 관점 차이의 뷰(View)를 제공하고 있다.

〈그림 34〉와 같이 특정 Branch로 이동할 수 있는 수단을 제공하는 Jump를 제공한다. Develop, master, origin/develop, origin/hotfix/1.0.1, 1.0.0 Tag, 1.0.1 Tag 에서 가장 마지막으로 했던 commit history를 보여준다.

그림 34_Jump

Ⅳ. Source Tree로부터 얻을 수 있는 장점

지금까지 세 번의 연재를 통해 GIT Flow를 활용한 효과적인 소스 형상 관리에 대해 살펴보았다.

소스 간의 충돌을 최소화해 원활한 소스 관리를 하고, SW 버전 별 유지보수가 용이해 효율적인 개발을 지원하기 때문에 GIT Flow를 통해 개발자들의 효율적인 업무 향상을 기대할 수 있을 것이다.

일반적으로 GIT를 사용하려면 터미널에서 작업하거나 언어별로 나누어진 개발(IDE)툴에 의존적인 플러그인(이클립스의 경우라면 EGit)을 써야 한다. 그러나 Source Tree를 사용하면 개발 툴에 독립적으로 개발이 가능하며, 가장 쉬운 UI 접근성을 제공하기 때문에 개발 시간을 줄일 수 있다.

또한 여러 명령어 또는 옵션을 하나씩 보면 일일이 접근하지 않아도 팝업창 또는 체크박스 옵션을 바탕으로 그 의미를 유추할 수 있어 사용이 훨씬 편리하다. Source Tree는 GIT Flow를 최적화하여 쓸 수 있는 다양한 기능을 제공하고 있고, GIT 프로젝트 리스트를 일목요연하게 관리할 수 있다.

게다가 무료(Free)로 이용할 수 있으니 적극적으로 활용해보기 바란다.

참고 자료

1. http://blog.sourcetreeapp.com/2012/08/01/smart-branching-with-sourcetree-and-git-flow/
2. http://www.sourcetreeapp.com
3. http://github.com
4. https://help.github.com/articles/generating-ssh-keys
5. http://git-scm.com/book/en/Git-on-the-Server-Generating-Your-SSH-Public-Key
6. https://answers.atlassian.com/questions/55442/sourcetree-license
7. http://blog.sourcetreeapp.com/2012/02/16/abandoning-the-mac-app-store/