2014. 2. 3. [제82호]

GIT Flow를 활용한 효과적인 소스 형상 관리
Part 1 : GIT Flow 는 무엇인가

소프트웨어공학센터 경영지원TF팀

Contents

정보통신산업진흥원 부설
nipa 소프트웨어공학센터

산업계에서는 단순히 GIT를 사용하기도 하지만 대다수의 대규모 프로젝트에서는 어느 정도 프로세스화된 GIT Branch 전략을 사용하여 개발하고 있다. 이 중에서 가장 널리 사용되고 있는 GIT Branch 전략 중의 하나인 GIT Flow를 살펴보고자 한다.

GIT Flow는 GIT 의 가장 큰 장점인 GIT Branch를 활용한다. 일반적으로 소프트웨어를 버전 단위로 릴리즈하는 유지보수 관점에서 GIT Flow는 매우 훌륭하다. GIT Flow는 배포관점의 형상 관리를 담당한다고 볼 수 있는데 원활한 소스코드 관리를 가능하게 하는 것이 핵심이다.

보통은 개발 중인 버전(Develop)과 이미 배포된 버전(Master)을 먼저 분리한 후 사용하는데, GIT Flow는 개발 중인 버전(Develop)을 바탕으로 중/단기 단위로 개발(Release, Feature)할 수 있도록 한다. 만약 치명적인 결함이 발생하더라도 이미 배포된 버전(Master)를 바탕으로 Hotfix 버전을 생성하여 빠른 패치를 진행하고, 개발 중인 버전(Develop)으로 소스 병합(Merging) 작업을 통해서 소스 형상관리를 최대한 활용할 수 있도록 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.

Ⅰ. GIT Flow 소개

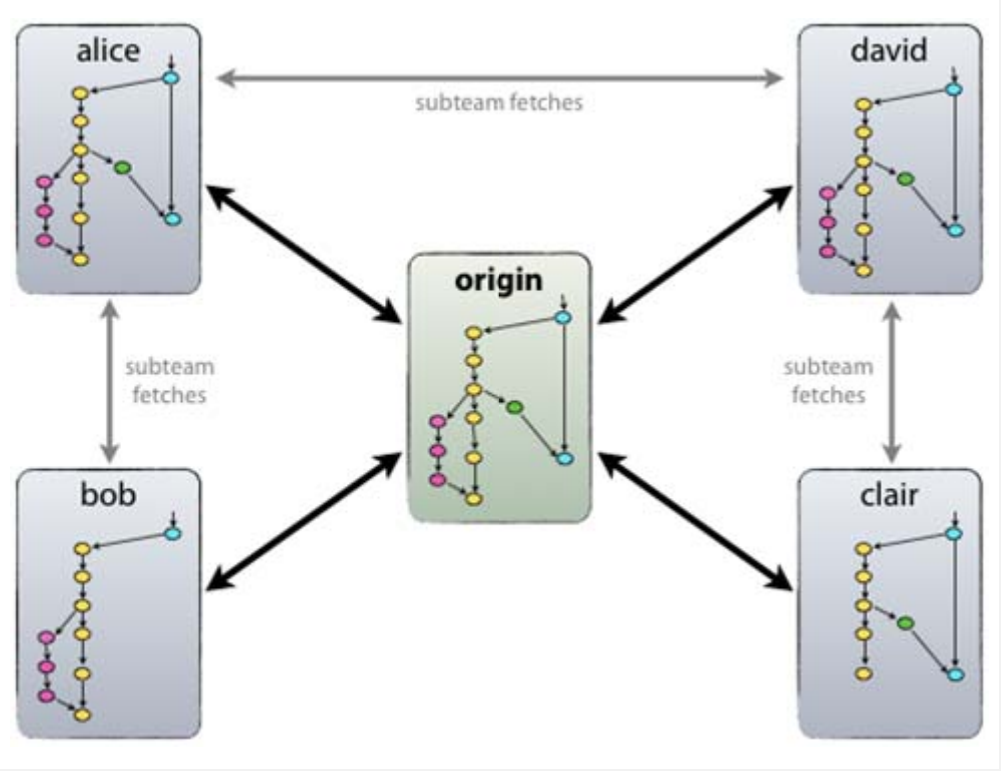
1. 개념

GIT은 로컬(Local)에서 다루기 쉽고 Branch를 자유롭게 생성할 수 있다. Branch 내에서 자유롭게 개발하고 여러 Branch 간에 Merge가 쉬울 뿐 아니라 Branch끼리 충돌하는 경우에도 부드러운 해결이 가능하다. 이런 점들이 제공되기 때문에 기존 형상관리 툴(Tool)인 SVN과 CVS와는 달리 소스를 최신 버전으로 유지하면서 개발자들이 편하게 사용할 수 있다.

〈그림 1〉은 여러 개발자가 GIT을 사용하여 origin이라는 원본 소스를 두고 여러 Branch에서 서로 상호 작용하는 것을 나타낸 그림이다. 그림을 보면 david와 alice가 origin 소스를 다운 받아 서버로부터 소스를 내려 받고 저장한다. alice의 소스를 bob이

내려 받아 작업 후 origin과 alice가 쓰는 Branch에 영향을 준다. 반면 david의 소스를 clair가 내려 받아 작업한 것을 origin과 david의 Branch에 영향을 주도록 하여 독립적이면서 유기적인 결합을 이끌어내는 특성을 갖도록 하고 있다.

그림 1_GIT Branch를 이용한 개발



출처: http://nvie.com/posts/a-successful-git-branching-model/

2. 역사와 트렌드

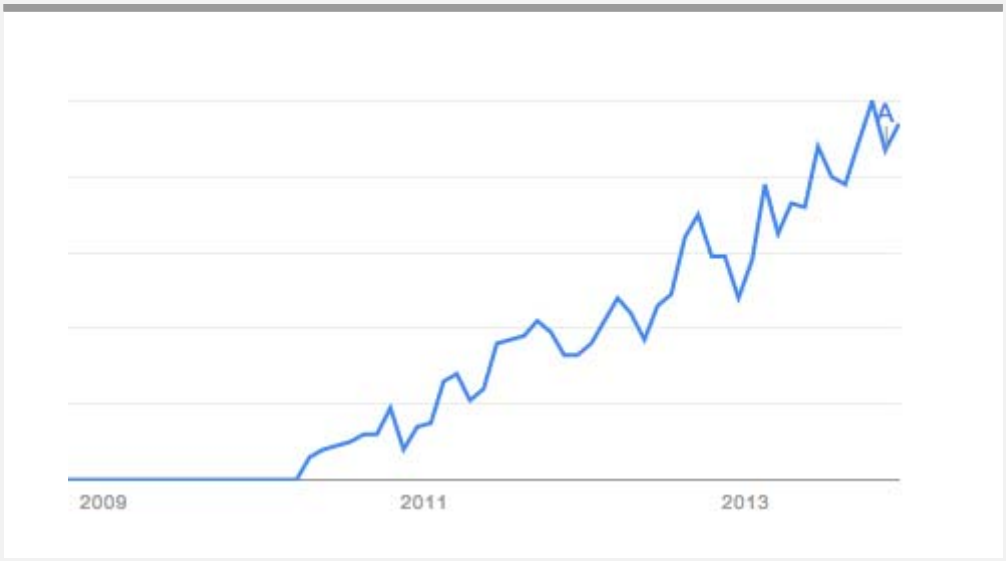
GIT의 가장 큰 장점인 Branch 전략을 잘 정립해서 GIT Branch를 활용한 GIT Flow 방식이 가장 많이 쓰이고 있다. GIT Flow는 네덜란드 출신의 Vincent Driessen라는 python 개발자가 공유한 Branch 전략으로 알려져 있다.[1] Vincent Driessen가 GIT Flow 방법론을 공유하면서 많은 사람들이 쓰기 시작했고, 현재 37명의 Contributer가 잘 유지시키고 있다. 관련 소스는 GitHub(https://github.com/nvie/gitflow)에서 확인해 볼 수 있다.

〈그림 2〉의 구글 트렌드가 보여주듯이 빠른 속도로 구글 검색이 이루어지고 있다. 이러한 가파른 관심도 상승 추세를 본다면 향후에는 더 많은 사람들이 GIT Flow를 찾아보

1) http://nvie.com/posts/a-successful-git-branching-model/

고, 이용할 것으로 예상할 수 있다.

그림 2_GIT Flow 구글 트렌드 – 시간 흐름에 따른 관심도 변화



출처: http://www.google.com/trends/explore#q=git%20flow

〈그림 3〉을 보면 미국 실리콘 밸리를 중심으로 GIT Flow 가장 많이 사용되고 있음을 보여준다.

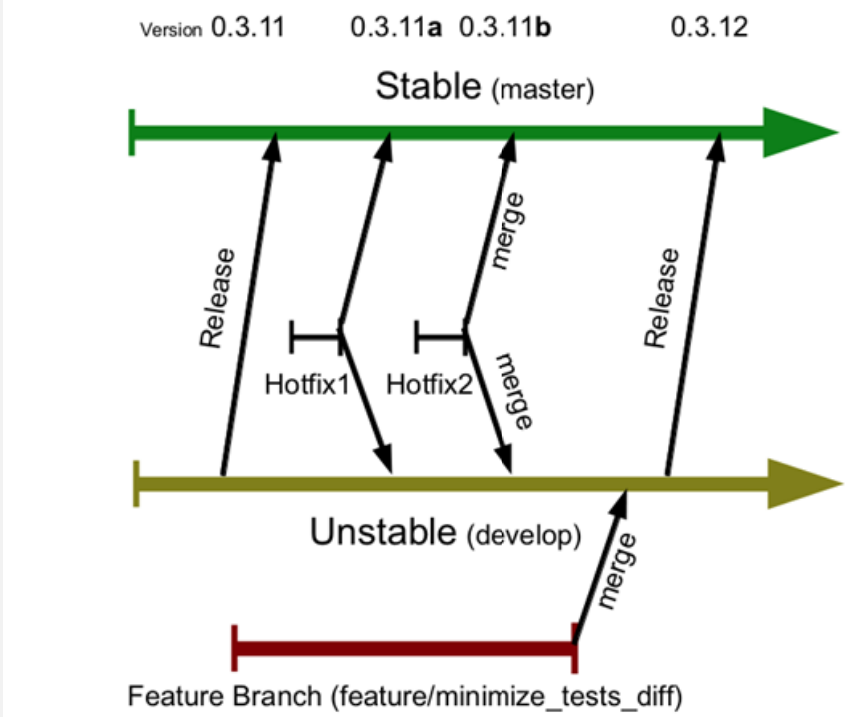
그림 3_GIT Flow 구글 트렌드 – 지역 관심도

출처: http://www.google.com/trends/explore#q=git%20flow

II. Branch 전략

1. GIT Flow 간단 설명

그림 4_GIT Flow의 대략적인 설명도



출처: http://nuitka.net/posts/nuitka-git-flow.html

〈그림 4〉는 GIT Flow를 쉽게 이해할 수 있도록 도식화 한 것이다. 처음에는 Master로 불리우는 안정적인 Branch(혹은 현재 배포된 버전)를 기반으로 Develop Branch가 존재한다. Master는 가장 안정적이지만 Develop은 계속 패치(patch)와 버전 업(upgrade)을 위한 개발이 일어난다. 버전 업을 위한 개발(Release) 진행을 완료하거나 긴급 버그 패치를 위한 개발(Hotfix)을 완료 후에 Master와 Develop Branch에 Merge(소스 합치기)를 진행한다.

특정 기능을 위한 개발(Feature) 작업 후 Develop Branch에 Merge 한다.

2. 중요 Branch 전략

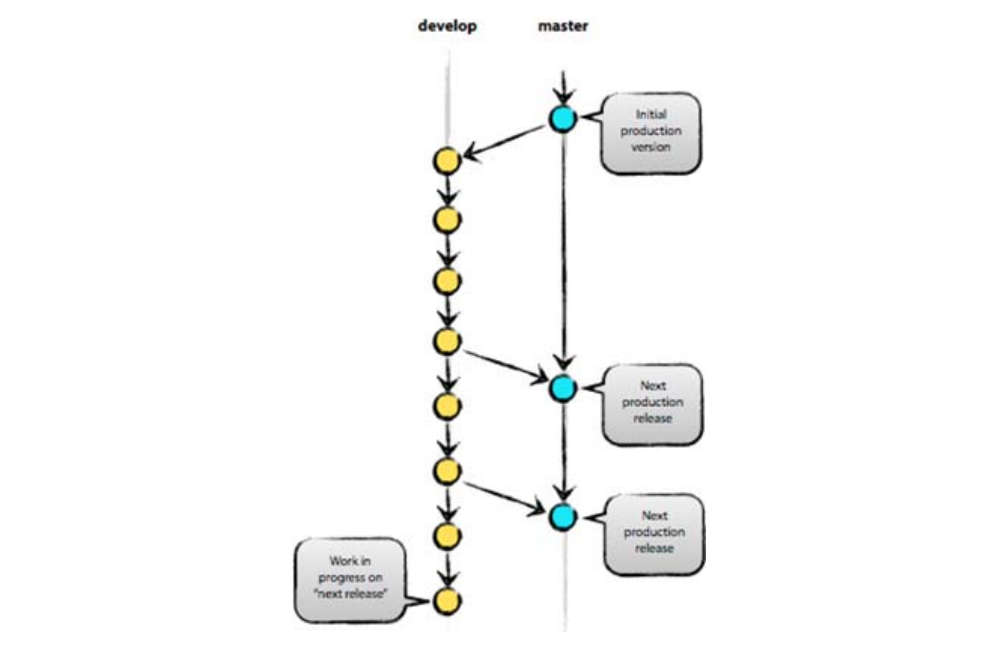
1) 초기화

– git fow init

GIT Flow Branching 모델로 GIT Repository를 변경한다. GIT Flow를 쓸 수 있도록 초기화 하는 과정이다.

〈그림 5〉와 같은 방식으로 개발할 수 있는데, GIT Flow를 초기화하면 Master와 Develop Branch가 생성된다. Develop Branch에서 기능을 추가하고 배포를 위해서 Master Branch에 Merge 한다.

그림 5_Master, Develop Branch 관계도



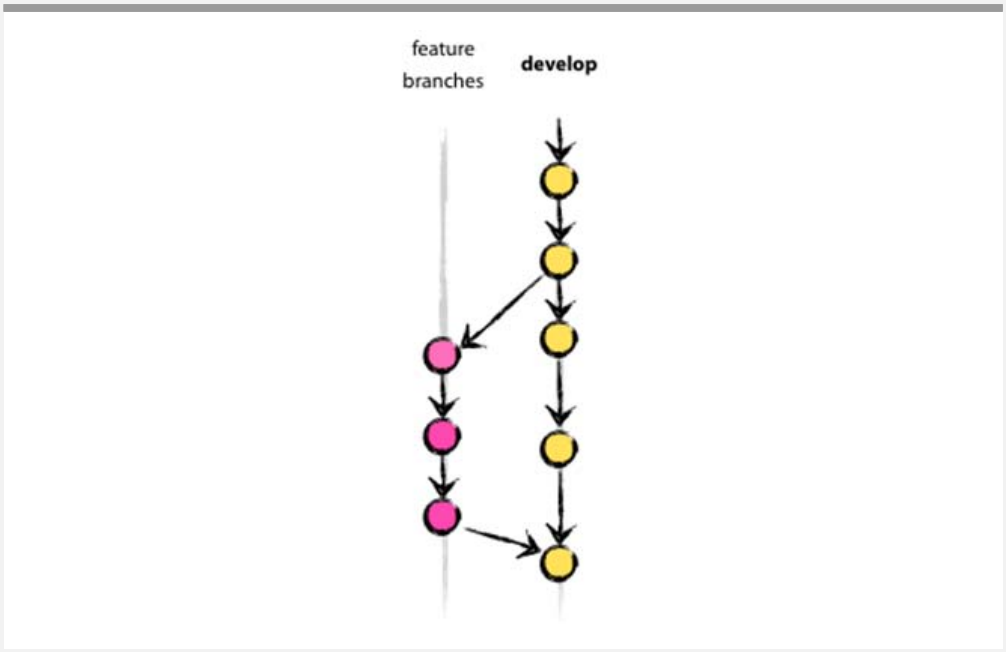
출처: http://nvie.com/posts/a-successful-git-branching-model/

2) Feature(기능)

Feature는 Release를 할 때 기능을 위한 Branch이다. 일반적으로 개발자의 로컬 PC에만 존재한다. 그러나 여러 개발자의 공유가 필요할 때는 GIT 원격 서버에 저장한다.

Develop Branch 기반이고, 작업 후에는 Develop Branch에 Merge한다. 그리고 Develop Branch 이외의 버전과는 Merge하지 않는다. Develop Branch에 Merge 하면 로컬에 있는 Feature Branch는 삭제된다.

그림 6_Feature Branch 와 Develop Branch간의 연관도



출처: http://nvie.com/posts/a-successful-git-branching-model/

〈그림 6〉은 Feature Branch를 Develop기반으로 생성하고, Develop Branch의 작업과 Feature Branch 간의 작업을 병렬적으로 작업하고 Merge할 수 있음을 알려준다.

- git flow feature

현재 존재하는 Feature Branch List를 출력한다.

- git flow feature start 〈name〉

Develop Branch에 기반 한 name으로 지정한 Feature Branch가 생성하고 그 Branch로 전환한다.

- git flow feature finish 〈name〉

Feature Branch를 Develop Branch로 Merge하고, Develop Branch로 전환한다.

- git flow feature publish 〈name〉

Feature Branch를 원격 서버에 Publish 하여 다른 개발자들도 같이 사용할 수 있도록 한다.

- git flow feature pull 〈name〉

다른 개발자가 Publish한 Feature Branch를 가져오고 변경을 추적한다.

- git flow feature diff 〈name〉

해당 Feature Branch와 Develop Branch 간의 소스를 비교(Merge-Base기준) 한다.

3) Release(출시)

다음 버전 또는 새로운 제품 배포를 위한 Branch이다. 버전 업을 할 때는 사소한 버그와 다양한 기능이 추가된다. Develop Branch 기반이고 작업이 완료되면 Develop와 Master 두 개의 Branch로 Merge한다. 배포 대상이 되는 Feature Branch는 Release Branch를 Develop Branch 에 Merge하기 전에 Release Branch에 Merge 되어야 한다. Release Branch에 반영되지 말아야 할 Feature는 Develop Branch에 Merge되어서는 안 된다. Release Branch의 Bug Fix는 Release Branch 안에서 해결한다.

- git flow release start 〈name〉

Develop Branch를 기반 한 name으로 지정한 Release Branch를 생성한다.

Release Branch 이름은 'release/v1.0.0'이라 많이 사용하고 있지만 편의를 위해서 'release-1.0.0'라고 쓰기도 한다.

- git flow release publish 〈name〉

Release Branch 생성 후에는 다른 개발자들에게 Release Branch 소스를 공유하여 commit를 허용한다. GIT Flow Feature Publish와 비슷한 기능이다.

- git flow release finish 〈name〉

다음 4가지가 순차적으로 일어난다.

① Relase Branch를 Master Branch에 Merge 한다.

② 그리고, Release 이름으로 Tag가 된다.

③ 그 다음에는 Release Branch를 Develop Branch로 Merge 한다.

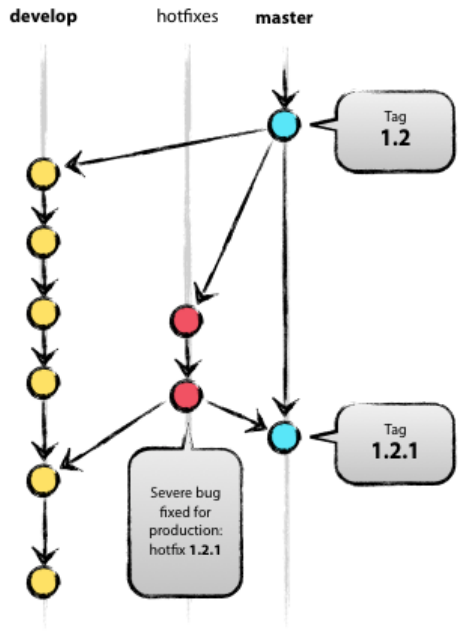
④ 마지막으로는 그 Release Branch가 삭제된다.

그러나 Tag가 원격 서버에 저장이 되지 않기 때문에 따로 'git push —tags'를 사용하여 Tag 들을 원격 서버에 Push하도록 한다.

4) Hotfix(긴급패치)

Hotfix는 현재 배포된 서비스 또는 제품에 문제 또는 중요 버그가 생겨서 긴급 버그 패치를 위한 Branch로 사용된다. Master Branch를 기반으로 표기된 Tag로부터 Branch를 따게 된다. Hotfix Branch 에서 긴급 패치를 진행한 Master에 Merge하고 버전을 Tag로 남긴다. Hotfix가 끝나면 Hotfix Branch는 삭제된다.

그림 7_Hotfix Branch 이용



출처: http://nvie.com/posts/a-successful-git-branching-model/

〈그림 7〉은 Hotfix Branch의 의미를 알려준다. 1.2에 critical bug가 발생되어 긴급하게 패치를 진행하고 테스트가 완료되면 Master Branch에 Merge 한 후, 1.2.1로 Tag를 생성한다. 그 후에 Develop Branch에도 Merge를 적용하여 똑같은 Bug가 재현되지 않도록 한다.

- git flow hotfix start 〈version〉

Master Branch를 기반으로 Hotfix version Branch를 생성한다.

예를 들면, GIT Flow Hotfix start 1.0.3와 같이 생성한다.

- git flow hotfix finish 〈version〉

Hotfix는 Master, Develop branch에 Merge 된다. Master Branch에 병합될 때는 Hotfix version이 Tag 된다.

5) 기타

GIT Flow에는 위에 설명한 Feature, Release, Hotfix Branch 외에 Support Branch가 존재한다. 생성과 종료 방법은 GIT Flow Hotfix나 Release처럼 똑같지만 거의 산업 현장에서는 많이 쓰이지 않는다.

Support는 Master Branch를 기반으로 만들어지는 것은 똑같지만 Hotfix와는 다르게 특정 목적을 위해 사용한다.

6) 개념도

그림 8_GIT Flow를 이용하여 개발할 때 흐름도

출처: http://www.codebeerstartups.com/2013/04/how-to-use-git-flow-in-your-projects

〈그림 8〉은 지금까지 설명한 Feature, Release, Hotfix를 이용하여 배포하는 것을 그린 그림이다. 분홍색 Feature Branch는 Develop Branch로 개발하다가 새로운 Feature를 생성, 개발 완료 후 Develop Branch에 완료한다. 그리고 녹색 Release Branch는 새로운 Release Branch를 Develop Branch 기반으로 생성, 개발이 완료되면 Develop와 Master Branch에 Merge한다. 이러한 흐름으로 바로 배포가 되었다. 빨간색 Hotfix Branch는 배포 후에 Critical한 버그가 발생하여 Hotfix Branch를 생성, 개발이 완료되면 Develop과 Master에 배포를 한다.

Part 1에서 살펴본 GIT Flow 이론을 기반으로 Part 2에서는 GIT Flow를 실제로 활용한 예제를 다뤄보도록 하겠다.

참고 자료

1. https://github.com/nvie/gitflow
2. http://nvie.com/posts/a-successful-git-branching-model/