2014. 2. 10. [제83호]

GIT Flow를 활용한 효과적인 소스 형상 관리
Part 2 : GIT Flow 실습과 활용 예제

소프트웨어공학센터 경영지원TF팀

Contents

Ⅰ. GIT Flow 소개
Ⅱ. Branch 전략
Ⅲ. 실제 사용 예제
Ⅳ. 결론

정보통신산업진흥원 부설
nipa 소프트웨어공학센터

Ⅲ. 실제 사용 예제

1. GIT Flow 사용 준비

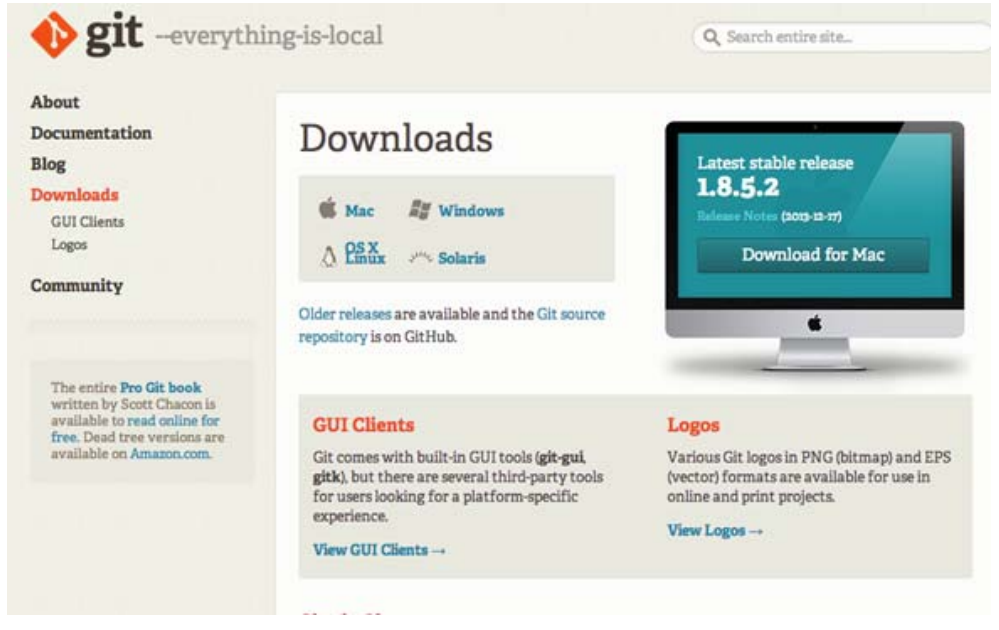
GIT Flow를 사용할 수 있도록 GIT 과 GIT Flow 설치를 진행하고, 무료 GIT Repository인 GIT Hub에 GIT Flow를 테스트 해볼 수 있는 GIT Repository를 생성한다.

1) GIT 설치

GIT 프로그램은 운영체제에 자동으로 설치되어 있지 않다. 〈그림 1〉의 다음 URL에 접속해서 운영체제별로 설치한다. http://git-scm.com/downloads

MAC OS X 개발자라면 링크에 접속해서 Mac 아이콘을 선택하고 dmg 설치 파일을 설치하고 GIT을 설치한다.

그림 1_GIT 설치 URL



출처: http://git-scm.com/downloads

GIT을 설치했다고 해서 GIT Flow를 바로 사용할 수는 없다.

```
$ git flow
git: 'flow' is not a git command. See 'git --help'.

Did you mean one of these?
reflog
show
```

GIT Flow를 설치해야 한다.

2) GIT Flow 설치

GIT Flow 는 MAC OS X, Linux(Unix), Windows까지 지원한다. 본인의 운영체제에 맞게 GIT Flow를 설치한다. 설치 주소는 https://github.com/nvie/gitflow/wiki/Installation에 있다.

만약 MAC OS X 를 사용하는 개발자라면, 상기 URL에서 MAC OS X 링크를 따라 들어가던지. https://github.com/nvie/gitflow/wiki/Mac-OS-X 에 바로 접속하여 설치할 수 있다.

현재 나와 있는 링크의 내용으로는 GIT Flow를 설치하려면 Homebrew나 MacPorts, Wget, Curl로 설치할 수 있다. 어느 것이든 상관없이 설치가 가능하므로, 개발자가 편한 방식으로 설치한다.

만약 Curl을 이용한다면, 아래와 같이 실행하면 설치가 된다.

```
curl -L -O https://raw.github.com/nvie/gitflow/develop/contrib/gitflow-installer.sh
sudo bash gitflow-installer.sh
```

https 주소에 연결된 github.com의 bash shell script를 다운받아 sudo 권한으로 실행시킨 결과는 다음과 같다. GIT Flow과 관련된 스크립트는 /usr/local/bin 디렉토리에 설치되는 것을 확인할 수 있다.

```
$ curl -L -O https://raw.github.com/nvie/gitflow/develop/contrib/gitflow-installer.sh
 % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
 0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0sudo bash
gitflow-installer.sh
100  2145  100  2145    0     0   2057      0  0:00:01  0:00:01 --:--:--  2058
$ sudo bash gitflow-installer.sh
Password:
### gitflow no-make installer ###
Installing git-flow to /usr/local/bin
Cloning repo from GitHub to gitflow
Cloning into 'gitflow'...
```

```
remote: Reusing existing pack: 1407, done.
remote: Total 1407 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (1407/1407), 623.29 KiB | 231.00 KiB/s, done.
Resolving deltas: 100% (689/689), done.
Checking connectivity... done
Updating submodules
Submodule 'shFlags' (git://github.com/nvie/shFlags.git) registered for path 'shFlags'
Cloning into 'shFlags'...
remote: Reusing existing pack: 454, done.
remote: Total 454 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (454/454), 130.79 KiB | 68.00 KiB/s, done.
Resolving deltas: 100% (338/338), done.
Checking connectivity... done
Submodule path 'shFlags': checked out '2fb06af13de884e9680f14a00c82e52a67c867f1'
install: gitflow/git-flow -> /usr/local/bin/git-flow
install: gitflow/git-flow-init -> /usr/local/bin/git-flow-init
install: gitflow/git-flow-feature -> /usr/local/bin/git-flow-feature
install: gitflow/git-flow-hotfix -> /usr/local/bin/git-flow-hotfix
install: gitflow/git-flow-release -> /usr/local/bin/git-flow-release
install: gitflow/git-flow-support -> /usr/local/bin/git-flow-support
install: gitflow/git-flow-version -> /usr/local/bin/git-flow-version
install: gitflow/gitflow-common -> /usr/local/bin/gitflow-common
install: gitflow/gitflow-shFlags -> /usr/local/bin/gitflow-shFlags
```

GIT Flow가 정상적으로 설치되었는지 터미널에서 확인한다. 'command not found' 결과가 아닌 Usage에 대한 결과가 나오면 정상적인 것이다.

```
$ git flow
usage: git flow <subcommand>

Available subcommands are:
   init      Initialize a new git repo with support for the branching model.
   feature   Manage your feature branches.
   release   Manage your release branches.
   hotfix    Manage your hotfix branches.
   support   Manage your support branches.
   version   Shows version information.

Try 'git flow <subcommand> help' for details.
```

3) GIT Hub 가입 및 GIT Repository 생성

GIT Flow 사용을 위한 GIT Repository를 생성해야 한다. 유명한 GIT Repository인 GIT Hub를 사용하면 된다.

http://github.com 가입/로그인 후, http://github.com/new 에서 GIT Flow를 테스트 할 github project를 생성한다. 그림과 같이 git-flow-test 라는 이름으로 프로젝트를 생성한다.

여기에서는 ksmark라는 계정을 만들고 테스트를 진행한다.

https://github.com/ksmark/git-flow-test 으로 생성된 것을 볼 수 있다.

4) GIT Repository 생성

```
$ git remote add origin https://github.com/ksmark/git-flow-test.git
$ mkdir git-flow-test
$ cd git-flow-test/
$ touch README.md
$ git init
Initialized empty Git repository in /development/git-flow-test/.git/
$ git add .
$ git commit -m 'first commit'
[master (root-commit) fefd8fb] first commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md

$ git push -u origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 217 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:ksmark/git-flow-test.git
* [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

https://github.com/ksmark/git-flow-test URL에 접속하면 commit/push했던 README.md 파일을 볼 수 있다. 또는 https://github.com/ksmark/git-flow-test/blob/master/README.md URL 에 접속하면 commit 했던 시간과 파일 정보를 볼 수 있다.

2. GIT Flow Branch 전략

1) GIT Flow용 Repository로 변경하기

지금까지 작업한 것은 GIT Repository를 사용할 수 있는 환경까지 진행했다. GIT Flow Branch 전략의 가장 기본인 Master, Develop Branch 두 개를 사용할 수 있는 환경, GIT Flow Branch 생성 시 GIT Flow Branch 전략에 맞게 생성되고 삭제되어야 한다. GIT flow init을 실행하여 naming에 대한 정책을 지정할 수 있다.

```
$ git flow init

Which branch should be used for bringing forth production releases?
   - master
Branch name for production releases: [master]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
```

GIT Flow 환경이 되었는지 GIT Branch 명령어를 통해 확인한다. Develop Branch의 결과를 보면 Master Branch외에 Develop Branch가 하나 생겼고, 현재 Develop Branch로 작업 중이라고 알려준다.

Develop Branch를 GIT Repository에 생성하여 다른 개발자들도 모두 볼 수 있도록 한다.

```
$ git branch
* develop
  master

$ git push origin develop
Counting objects: 1, done.
Writing objects: 100% (1/1), 249 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To git@github.com:ksmark/git-flow-test.git
* [new branch]      develop -> develop
```

.git/config 파일 안에는 GIT Flow 관련 처리 내용이 저장되어 있다.

```
        [branch "master"]
            remote = origin
            merge = refs/heads/master
[gitflow "branch"]
            master = master
            develop = develop
[gitflow "prefix"]
            feature = feature/
            release = release/
            hotfix = hotfix/
            support = support/
            versiontag =
```

2) Develop Branch 작업과 Push

개발 PC상의 Develop Branch 상에서 pom.xml 파일을 추가하고 commit 후 push 한다.

```
$ touch pom.xml
$ git status
# On branch develop
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#         pom.xml
nothing added to commit but untracked files present (use "git add" to track)
$ git add .
$ git commit -m 'pom.xml added.'
$ git push
```

3) GIT Flow Feature 사용하기

GIT Flow Feature 는 Develop Branch 기반 위에 만들어져 기능 단위의 개발에 유용하게 사용한다. login-ui 라는 이름으로 feature를 생성(git flow feature start 'login-ui')하고 나면 로컬 GIT Branch는 자동으로 해당 Branch로 이동한다.

```
$ git flow feature start 'login-ui'
Switched to a new branch 'feature/login-ui'

Summary of actions:
- A new branch 'feature/login-ui' was created, based on 'develop'
- You are now on branch 'feature/login-ui'
```

```
Now, start committing on your feature. When done, use:

     git flow feature finish login-ui

$ git branch
 develop
* feature/login-ui
 master
```

login-ui 관련 작업을 위해서 src 디렉토리에 3개의 png 파일과 login.js 파일을 추가하고 login-ui branch를 Develop Branch로 Merge한다. (git flow feature start 'login-ui') feature/login-ui branch 에서 작업했던 내용은 Develop Branch로 Merge되고 Develop Branch는 삭제된다. 그리고 로컬 GIT Branch는 Develop로 이동한다.

```
$ mkdir src
$ touch src/login-ui1.png
$ touch src/login-ui2.png
$ touch src/login-ui3.png
$ touch src/login.js

$ git flow feature finish
Switched to branch 'develop'
Already up-to-date.
Deleted branch feature/login-ui (was b3e057c).

Summary of actions:
- The feature branch 'feature/login-ui' was merged into 'develop'
- Feature branch 'feature/login-ui' has been removed
- You are now on branch 'develop'

$ git branch
* develop
 master

$ git status
# On branch develop
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
```

```
#           src/
nothing added to commit but untracked files present (use "git add" to track)
```

Develop Branch로 Merge 된 내용을 작업한 내용 모두를 Local Develop Branch에 commit 후 GIT Respository에 Push 하여 모두 반영한다.

```
$ git add .
$ git commit -m 'login-ui commited.'
[develop 85c849b] login-ui commited.
4 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 src/login-ui1.png
create mode 100644 src/login-ui2.png
create mode 100644 src/login-ui3.png
create mode 100644 src/login.js
$ git push
```

4) GIT Flow Release 사용하기

1.0.0 배포를 위해서 1.0.0 Release Branch를 만든다. (git flow release start 1.0.0)

로컬 GIT Branch는 Develop에서 release/1.0.0 으로 이동한다. 다른 개발자와 함께 개발을 동시에 진행해야하기 때문에 서버에도 Branch를 생성한다.(git push origin release/1.0.0)

```
$ git flow release start 1.0.0
Switched to a new branch 'release/1.0.0'

Summary of actions:
- A new branch 'release/1.0.0' was created, based on 'develop'
- You are now on branch 'release/1.0.0'

Follow-up actions:
- Bump the version number now!
- Start committing last-minute fixes in preparing your release
- When done, run:

     git flow release finish '1.0.0'

$ git branch
 develop
 master
* release/1.0.0
```

```
$ git push origin release/1.0.0
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 527 bytes | 0 bytes/s, done.
Total 5 (delta 1), reused 0 (delta 0)
To git@github.com:ksmark/git-flow-test.git
* [new branch]      release/1.0.0 -> release/1.0.0
```

배포 버전 1.0.0에서 적용할 작업 내용을 개발한다.

```
$ mkdir login
$ touch login/LoginController.java
$ touch login/LoginService.java
$ touch login/LoginRepository.java
$ git add .
$ git commit -m 'login logic added.'
[release/1.0.0 6a2e09d] login logic added.
3 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 login/LoginController.java
create mode 100644 login/LoginRepository.java
create mode 100644 login/LoginService.java
$ git push
```

기능 단위로 commit & push 을 진행한다.

```
$ touch login/LogOutController.java
$ git add .
$ git commit -m 'log-out logic added.'
[release/1.0.0 93529ca] log-out logic added.
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 login/LogOutController.java
$ git push
```

개발 작업이 모두 완료되면 마무리를 한다. (git flow release finish 1.0.0)

Release Finish 시에는 다음의 순서대로 차례로 일어난다.

- release/1.0.0 branch의 최근 소스가 Master Branch에 Merge
- 로컬에 tag 1.0.0 이 생성
- release/1.0.0 branch는 Develop Branch로 Merge 됨
- 로컬의 release/1.0.0는 삭제됨 (Git Repository에 push된 release/1.0.0은 그대로 존재)

```
$ git flow release finish 1.0.0
Switched to branch 'master'
Merge made by the 'recursive' strategy.
login/LogOutController.java | 0
login/LoginController.java  | 0
login/LoginRepository.java  | 0
login/LoginService.java     | 0
pom.xml                     | 0
src/login-ui1.png           | 0
src/login-ui2.png           | 0
src/login-ui3.png           | 0
src/login.js                | 0
9 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 login/LogOutController.java
create mode 100644 login/LoginController.java
create mode 100644 login/LoginRepository.java
create mode 100644 login/LoginService.java
create mode 100644 pom.xml
create mode 100644 src/login-ui1.png
create mode 100644 src/login-ui2.png
create mode 100644 src/login-ui3.png
create mode 100644 src/login.js
Switched to branch 'develop'
Merge made by the 'recursive' strategy.
login/LogOutController.java | 0
login/LoginController.java  | 0
login/LoginRepository.java  | 0
login/LoginService.java     | 0
4 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 login/LogOutController.java
create mode 100644 login/LoginController.java
create mode 100644 login/LoginRepository.java
create mode 100644 login/LoginService.java
Deleted branch release/1.0.0 (was 93529ca).

Summary of actions:
- Latest objects have been fetched from 'origin'
- Release branch has been merged into 'master'
- The release was tagged '1.0.0'
```

```
- Release branch has been back-merged into 'develop'
- Release branch 'release/1.0.0' has been deleted

$ git branch
* develop
  master
```

Merge된 Develop, Master Branch와 tag 1.0.0은 모두 로컬에 있다. 이를 모두 GIT Repository에 push하여 반영하여야 한다. tag와 함께 push한다.

```
$ git push

$ git push --tags
Counting objects: 1, done.
Writing objects: 100% (1/1), 160 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To git@github.com:ksmark/git-flow-test.git
* [new tag]           1.0.0 -> 1.0.0
```

5) GIT Hotfix 사용하기

GIT Flow Release 와 약간 비슷한 절차를 거친다. 만약 배포된 버전인 Master Branch에서 버그가 발생해서 코드를 고친 후 배포해야 할 상황에서 사용한다.

hotfix/1.0.1 branch에 긴급 패치를 진행한다. 로컬 GIT은 hotfix/1.0.1 branch로 자동으로 이동한다.

```
$ git flow hotfix start 1.0.1
Switched to a new branch 'hotfix/1.0.1'

Summary of actions:
- A new branch 'hotfix/1.0.1' was created, based on 'master'
- You are now on branch 'hotfix/1.0.1'

Follow-up actions:
- Bump the version number now!
- Start committing your hot fixes
- When done, run:

     git flow hotfix finish '1.0.1'
```

```
$ git branch
  develop
* hotfix/1.0.1
  master
```

Bug Fix 할 소스를 수정한 후, commit & push를 진행 후에 Hotfix를 Merge한다. (GIT Flow Hotfix Finish) 작업한 소스를 가지고 Review 후에 마무리를 짓는다.

```
$ vi login/LoginController.java
$ git add .
$ git commit -m 'typo-critical issue fixed.'
[hotfix/1.0.1 0356dcb] typo-critical issue fixed.
1 file changed, 1 insertion(+)
$ git push
```

Hotfix Branch 개발 작업이 모두 완료되면 마무리를 한다. (git flow hotfix finish 1.0.1)

Hotfix Finish 시에는 다음의 순서대로 차례로 일어난다.

- hotfix/1.0.1 branch의 최근 소스가 모두 Master Branch에 Merge 되고 개발자가 Merge Log를 작성
- tag는 1.0.1 이 생성되고 개발자가 Merge Log 작성
- Hotfix Branch는 Develop으로 Merge 되고 개발자가 Merge Log를 작성
- 로컬의 hotfix branch는 삭제됨 (GIT repository에 push된 hotfix/1.0.1은 그대로 존재)
- Develop Branch로 이동

```
$ git flow hotfix finish 1.0.1
Switched to branch 'master'
Merge made by the 'recursive' strategy.
login/LoginController.java | 1 +
1 file changed, 1 insertion(+)
Switched to branch 'develop'
Merge made by the 'recursive' strategy.
login/LoginController.java | 1 +
1 file changed, 1 insertion(+)
Deleted branch hotfix/1.0.1 (was 0356dcb).

Summary of actions:
- Latest objects have been fetched from 'origin'
- Hotfix branch has been merged into 'master'
- The hotfix was tagged '1.0.1'
```

```
- Hotfix branch has been back-merged into 'develop'
- Hotfix branch 'hotfix/1.0.1' has been deleted

$ git branch
* develop
  master
```

Git Repository에 tag 1.0.1과 Develop, Master Branch를 반영한다.

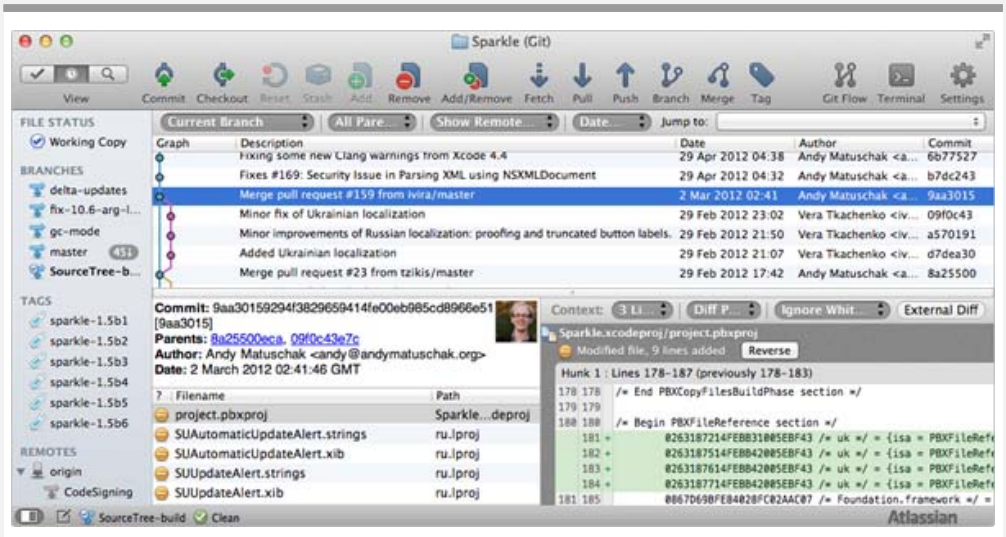
```
$ git push --tags

$ git push
```

3. UI Tool 참고

지금까지의 작업은 터미널에서의 작업으로 터미널에서의 명령어가 익숙하지 않을 때는 실수 할 수 있는 부분이 존재한다. 이를 위해서 〈그림 2〉의 Source Tree라는 오픈소스 툴을 활용하면 UI를 이용하여 GIT과 GIT Flow를 쉽게 이용 할 수 있다.

그림 2_Source Tree App 스크린 샷



출처: http://www.sourcetreeapp.com/

Ⅳ. 결론

지금까지 실습을 통해서 감을 잡아 보았다. Part 1에서 언급한 GIT Flow 도식 〈그림4〉를 다시 살펴보기를 권한다. GIT Flow를 사용하려면 Develop와 Master Branch를 활용해야 한다. Feature Branch 전략을 이용하면 Develop Branch에만 영향을 미치게 되고, Release Branch 전략을 이용하면 Master 와 Develop Branch에 차례로 영향을 미친다. Hotfix Branch 전략을 사용하면 Release Branch 전략과 비슷하게 Master와 Develop Branch에 영향을 미친다. Release Branch와 Hotfix Branch는 버전을 이용한 Branch이기 때문에 tag에 정보가 저장된다.

GIT Flow를 통해 배포 관점의 형상관리에 대한 실무 활용에 도움이 되기를 바란다.

참고 자료

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