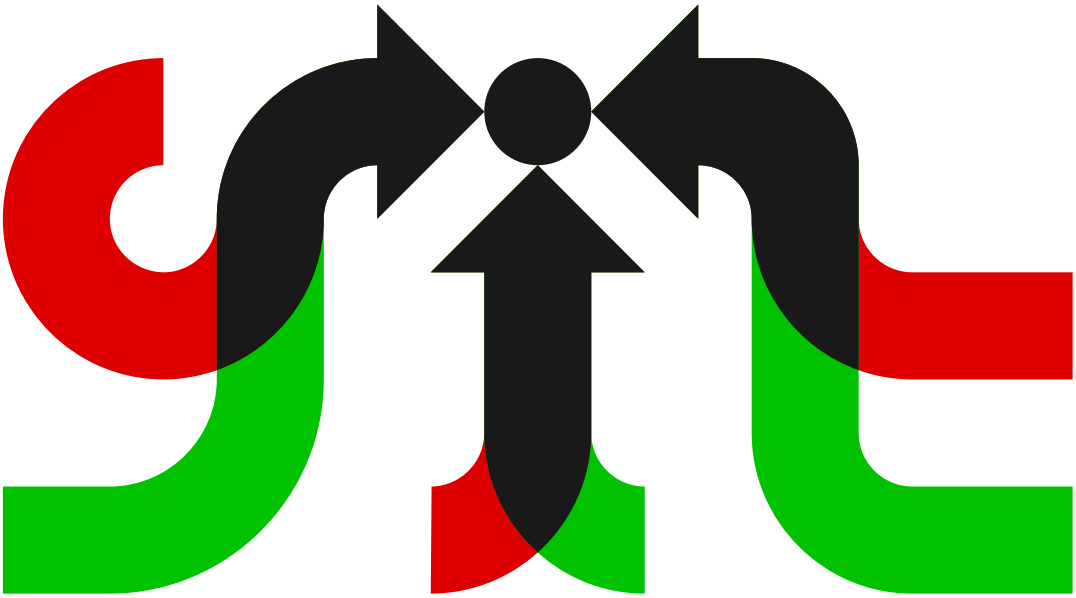
Git Guide



작성자 : 왕수용(wangsy@wangsy.com)

소속 : (주)민트기술 (mintech.kr)

2011. 12. 9

revision #1

개요

git 를 시작하기 전, 알면 좋은 몇가지 개념을 소개한다.

git 는 분산형

git 가 기존의 svn 과 구분되는 가장 큰 특징은 분산형이라는 것이다. 이것이 무슨 뜻이냐 하면, svn 의 경우 중앙 리포지토리에서 관리된다. 그래서, 모든 커밋이 중앙에 집중이 되고, 이것이 revision 번호로 관리가 된다. 중앙의 특정 revision 번호가 특정한 소스의 상태를 대변할 수 있는 것이다.

반대로, git 의 경우, 각각 분산된 곳에서 따로 따로 리포지토리가 관리된다. 그래서, 특정 revision 번호라는 것이 존재할 수 없다. 나는 나대로, 너는 너대로 따로 리포지토리가 관리된 상태에서 서로 머지를 할 수도 있고, 다시 분리할 수도 있다. revision 번호라는 것은 엄격한 순서가 관리되는데, 이 같은 분산형에서는 불가능한 일이다.

그래서, git 에서는 40자의 HEX CODE 를 commit ID 값으로 쓴다. 즉, 완전히 고유한 (동일한 값이 나올 확율이 거의 없는) 값을 각 커밋마다 부여한다. 그리고, 그 코드 값으로 각 커밋 단위를 부른다. 순서는 상관없다.

git 는 네단계로 관리를 한다.

untracked <=> tracked <=> staged <=> commited <=> pushed

프로젝트 시작하기

서버로 부터 가져오면서 시작하기

```
$ git clone ssh://user@server.com/git/project.git
$ git clone git://user@server.com/git/project.git
```

로컬에서 관리하면서 시작하기

```
$ mkdir project
$ cd project
```

```
$ git init
```

공유지점을 만들면서 시작하기

```
$ git init --bare project.git
```

프로젝트 진행 흐름 따라잡기

서버에서 생성되어 있는 브랜치 보기

```
$ git branch -r
origin/HEAD -> origin/master
origin/development
origin/master
```

서버에서 생성되어 있는 브랜치 가져오기

```
$ git checkout -b <new branch name> <server branch name>
$ git checkout -b development origin/development
```

내가 최초로 서버에 브랜치 만들기

```
$ git push <server name> <local branch>:<remote branch>
$ git push origin master:development
```

내가 clone 해 온 서버는 자동으로 서버의 이름으로 origin 이란 이름을 가진다. 아래 명령을 해 보면, 현재의 리포지토리와 연결된 서버의 이름의 목록이 출력 된다.

```
$ git remote
origin
```

또한 최초로 만들어진 branch 의 이름은 master 이다.

서버에서 진행된 상황 가져오기

```
$ git pull
```

새로운 기능 추가하기

branch 간 이동하기

```
$ git branch
development
* master
$ git checkout development
```

새로운 기능 추가를 위한 branch 만들기

```
$ git checkout -b feature-great-new development
```

새로 만든 branch 서버에 올리기

```
$ git push <server name> <local branch>:<remote branch>
$ git push origin feature-great-new:feature-great-new
```

추가한 기능 공유하기

일단 새로운 기능을 구현한다.

```
$ edit new-feature.c
```

그리고, 구현이 완료가 되면, 커밋한다.

```
$ git commit -a -m “Log Message”
```

그러면, 로컬 리포지토리에 커밋이 기록된다. 그리고, 서버에 반영하려면,

```
$ git push
```

새로운 기능 추가 마무리

구현이 마무리가 되면, development branch 에 merge 해 준다.

```
$ git checkout development
$ git merge feature-great-new
```

그리고, development branch 에 완전히 적용이 됐다면, feature-great-new 브랜치는 없애준다.

```
$ git branch -d feature-great-new
$ git push origin :feature-great-new
```

수정한 내용 취소하기

```
$ git checkout .
```

설정하기

내 정보 설정

```
# personalize these with your own name and email address
git config --global user.name "Sooyong Wang"
git config --global user.email "wangsy@wangsy.com"
```

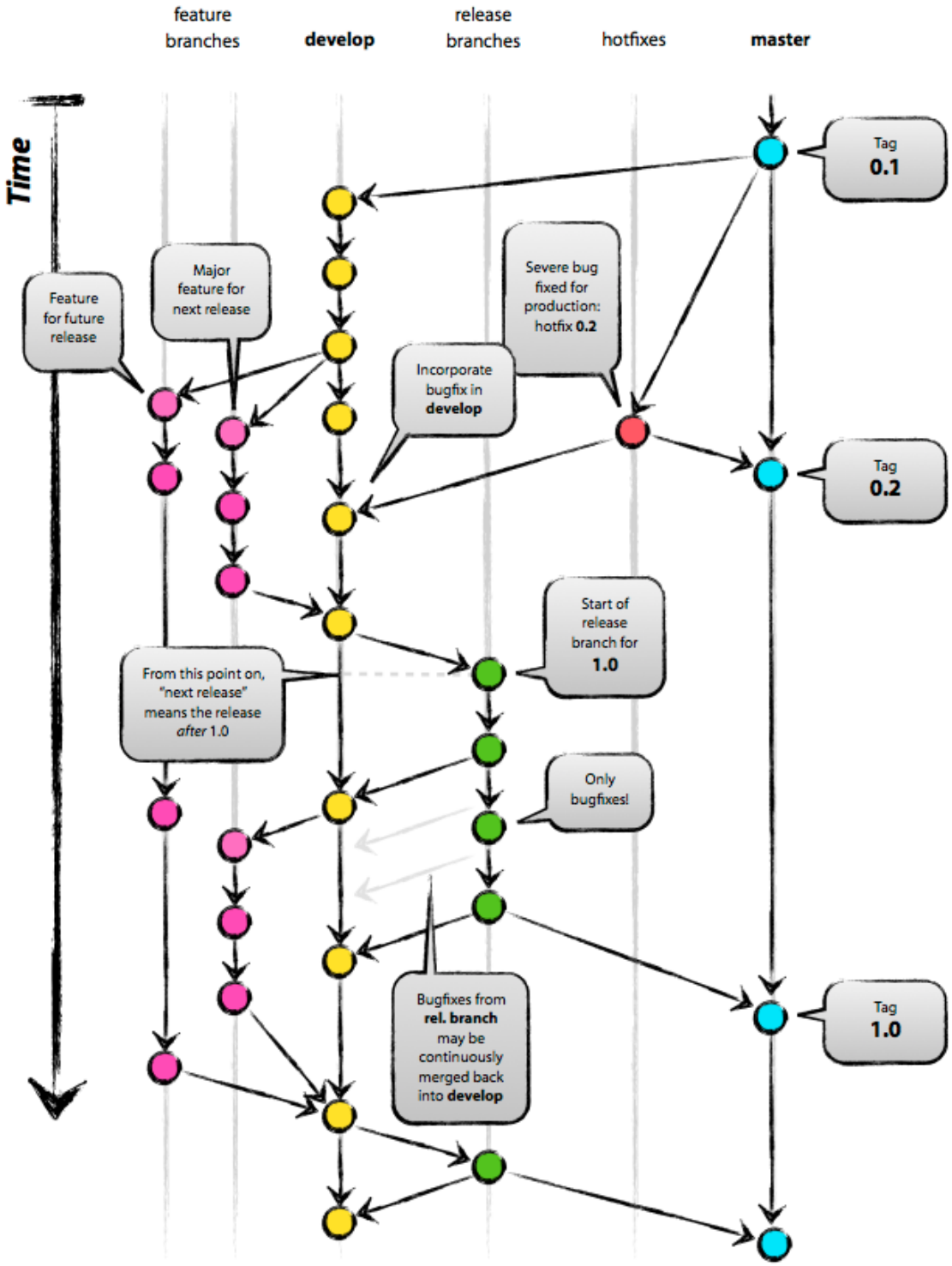
color 설정

```
$ git config --global color.ui auto
```

alias 설정

```
# shortcut aliases
$ git config --global alias.st status
$ git config --global alias.ci commit
$ git config --global alias.co checkout
```

Git-Flow

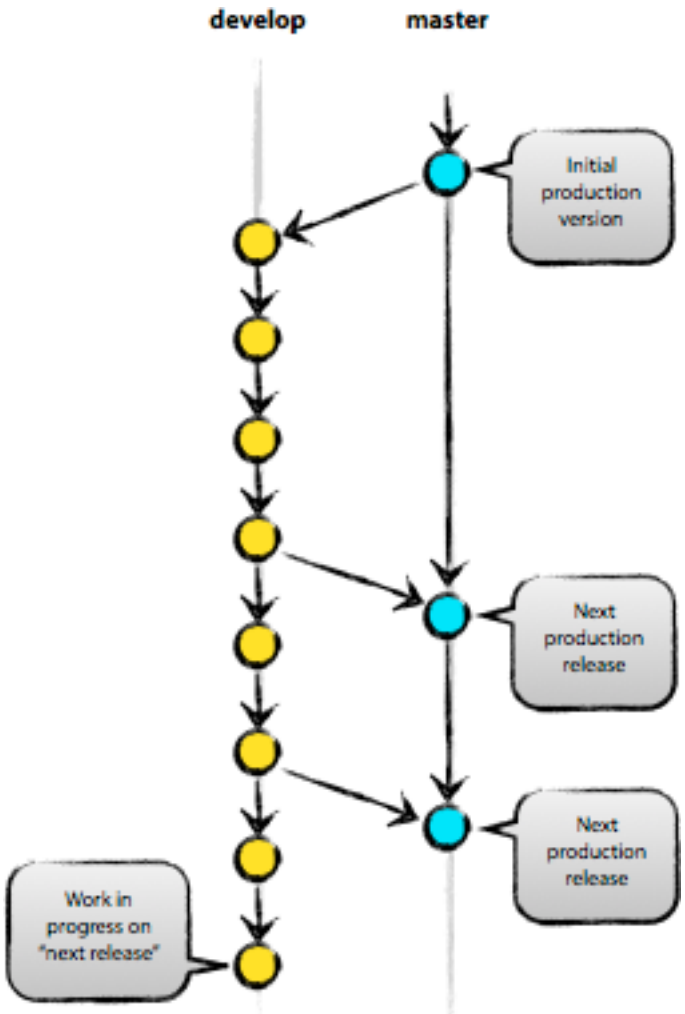

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

git-flow 는 빈센트 드리센이란 분이, 브랜치를 살 써서, 프로젝트의 리포지토리를 관리하는 법을 기술하고, git 확장 파일까지 만들었다. 이 확장파일을 설치하면, git 를 더 일관적으로 사용할 수 있다.

약간의 개념

master 과 develop 브랜치

origin 의 master 브랜치는 실제 최종 릴리즈된 소스코드만을 관리한다. 개발중의 코드는 모두 develop 브랜치에 커밋한다. 즉, 처음 master 브랜치에서 develop 브랜치를 만든 다음, 개발중 계속 develop 브랜치에 커밋을 하고, 최종 릴리즈가 된 다음, master 브랜치에 머지하는 방식이다.



feature 브랜치

feature 브랜치는 항상 develop 로 부터 브랜치를 만들며, 완료후에는 다시 develop 브랜치로 머지 된다.

특정 새로운 기능을 구현 할 때, feature 브랜치를 만들고, 다 만들고 나면, 이 브랜치를 제거해 준다.

일반적으로 feature 브랜치는 origin 에서는 관리하지 않는다.

release 브랜치

release 브랜치는 develop 으로부터 브랜치를 만들고, 나중에 develop 과 master 브랜치로 머지 된다. develop 브랜치에서 웬만한 기능 구현이 완료 되었을때, release 브랜치를 만든다. 그리고, 버젼명, 빌드 날짜 등을 수정하고, 테스트 중 오류를 수정 한다. 모든 것이 완벽하면, release 브랜치는 제거되면서, develop 과 master 로 머지 시킨다.

hotfix 브랜치

hotfix 의 경우, 현재 릴리즈 되어 있는 소스코드, 즉 master 상의 오류를 긴급 수정할 때 사용한다. master 로 부터 브랜치 하여, 긴급 수정하고, 테스트 한 이후, 문제가 없으면, master 와 develop 으로 머지 시킨다.

설치하기

Mac 에서는 두가지 설치방법이 있다.

homebrew 를 사용하는 경우

```
$ brew install git-flow
```

port 를 사용하는 경우

```
$ port install git-flow
```

시작하기

기존 프로젝트에서 git flow 를 적용하기 시작하려면, 아래와 같은 명령을 수행해 준다.

```
$ git flow init
```

그럼 많은 것을 물어보는데, 대부분 기본값을 사용하면 충분한다.

```
No branches exist yet. Base branches must be created now.
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? []
```

본격 프로젝트 진행하기

일반적으로 기본적인 구현 진행은 develop 브랜치에서 진행한다. 하지만, 완전히 새로운 큰 기능 구현을 시작한다면, 아래와 같은 명령을 사용한다.

```
git flow feature
git flow feature start <name> [<base>]
git flow feature finish <name>
```

위에서 <base> 는 develop 브랜치의 특정 commit ID 를 뜻한다. feature 를 finish 하면 해당 브랜치가 삭제가 된다.

만일 릴리즈 과정을 시작한다면, 아래의 명령을 사용한다.

```
git flow release
git flow release start <release> [<base>]
git flow release finish <release>
```

마찬가지로, <base> 는 develop 브랜치의 commit ID 를 말한다. release 가 finish 되면, master 브랜치에 머지가 된다.

릴리즈 후, 수정사항이 발생할 경우, hotfix 브랜치를 시작한다.

```
git flow hotfix
git flow hotfix start <release> [<base>]
git flow hotfix finish <release>
```

위에서 <base> 는 master 브랜치의 특정 commit ID 를 뜻한다.

gitosis

설치

설치는 리눅스의 경우 배포판마다 조금씩 차이가 있으니, 해당 배포판에 따라 설치한다.

관리자 입장

관리자 입장에서는 먼저, 설치를 마치고, 서버로 부터 gitosis-admin.git 리포지토리를 clone 해 준다.

```
$ git clone git@gitserver:gitosis-admin.git
```

`gitosis.conf` 파일을 편집해 준다. 그리고, 서버로 push 해 주면, 해당 내용이 반영된다.

새로운 프로젝트 추가

새로운 리포지토리가 추가되면, `gitosis.conf` 파일을 아래와 같이 추가해 준다.

```
[group committer]
members = member1 member2 member3

[group mobile_project]
readonly = second_project
members = member1 member2

[group web_project]
writable = third_project fourth_project fifth_project
readonly = sixth_project
members = @committer member4
```

그리고, 기존의 프로젝트를 올린다.

```
$ git remote add origin git@gitserver:repository.git
$ git push origin master
```

새로운 사용자 추가

첫째, 사용자로부터, public key 파일을 받아서, `keydir` 디렉토리에 복사해 준다. 이때, public 키 파일의 이름이 중요하다. public key 파일의 이름을 <사용자이름>.pub 형식으로 써 주고, 이때 사용한 <사용자이름>을 `gitosis.conf` 파일의 `members` 필드에 들어가는 이름과 반드시 일치해야 한다.

사용자 입장

먼저 `ssh-keygen` 을 이용해서, private key, public key 를 만들어 준다.

```
$ ssh-keygen -t rsa
```

이미 있을 경우도 있다. 아래 폴더를 확인해 본다.

```
$ cd ~/.ssh
$ ls
```

`id_rsa` 파일(private key)과 `id_rsa.pub` 파일(public key)이 있으면 된다. 혹시 둘 중 하나만 있다면, 아래 명령을 통해서 나머지를 만들어 낼 수 있다.

```
$ ssh-keygen -e
```

여기서 public key 파일을 열었을 때, 마지막 부분이 사용자의 이메일 주소이다. 이 부분을 git 에서 사용하는 이메일 주소로 수정해 준다.

관리자에게 이 public key 파일을 전달해 주면, 관리자가, 특정 리포지토리에 권한을 등록해 주면, 그 다음부터는 접근이 가능하다.

못다한 이야기들

- gerrit
- jenkins