자주 사용하는 기초 Git 명령어 정리하기

[Go to the profile of 박성룡 ( Andrew park )](https://medium.com/@pks2974?source=post_header_lockup)

[박성룡 ( Andrew park )](https://medium.com/@pks2974)Follow

Oct 13, 2017

제가 서비스를 개발하면서 자주 사용하던 git 명령어를 정리해보려고 합니다.



**들어가기전에**

1. 요약
2. Git이란 무엇인가?
3. Git 설치하기
4. 다른 개발자의 코드 받아오기
5. Branch 관리하기
6. Commit 내용 취소하기
7. 코드 잠깐 저장하고, 다른 브랜치로 이동하기

**요약**

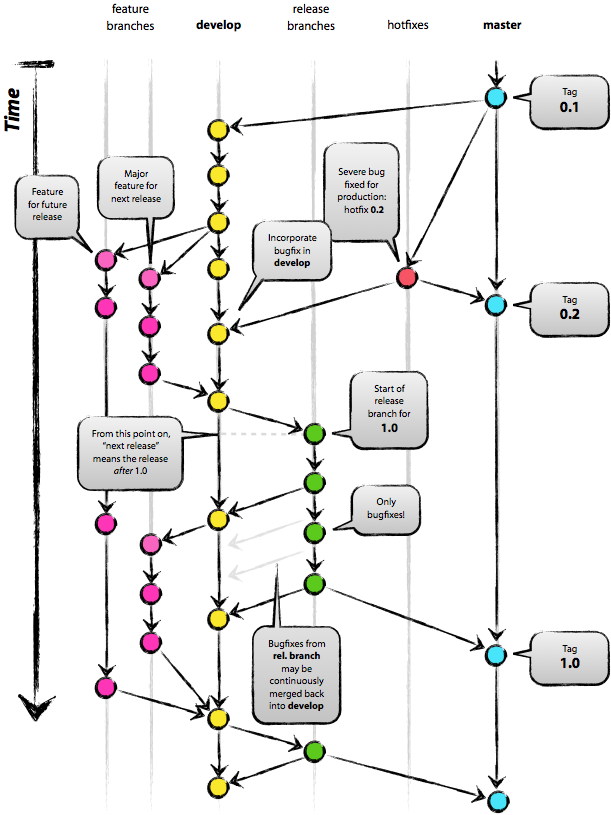
* git init : git 생성하기
* git clone git\_path : 코드가져오기
* git checkout branch\_name : 브랜치 선택하기
* git checkout -t remote\_path/branch\_name : 원격 브랜치 선택하기
* git branch branch\_name : 브랜치 생성하기
* git branch -r : 원격 브랜치 목록보기
* git branch -a : 로컬 브랜치 목록보기
* git branch -m branch\_name change\_branch\_name : 브랜치 이름 바꾸기
* git branch -d branch\_name : 브랜치 삭제하기
* git push remote\_name — delete branch\_name : 원격 브랜치 삭제하기 ( git push origin — delete gh-pages )
* git add file\_path : 수정한 코드 선택하기 ( git add \* )
* git commit -m “commit\_description” : 선택한 코드 설명 적기 ( git commit -m “내용”)
* git push romote\_name branch\_name : add하고 commit한 코드 git server에 보내기 (git push origin master)
* git pull : git서버에서 최신 코드 받아와 merge 하기
* git fetch : git서버에서 최신 코드 받아오기
* git reset — hard HEAD^ : commit한 이전 코드 취소하기
* git reset — soft HEAD^ : 코드는 살리고 commit만 취소하기
* git reset — merge : merge 취소하기
* git reset — hard HEAD && git pull : git 코드 강제로 모두 받아오기
* git config — global user.name “user\_name ” : git 계정Name 변경하기
* git config — global user.email “user\_email” : git 계정Mail변경하기
* git stash / git stash save “description” : 작업코드 임시저장하고 브랜치 바꾸기
* git stash pop : 마지막으로 임시저장한 작업코드 가져오기
* git branch — set-upstream-to=remote\_path/branch\_name : git pull no tracking info 에러해결

**Git이란 무엇인가?**

Git이란 souce 관리를 위한 분산 버전 관리 시스템입니다. 최초로 리눅스 토발즈가 리눅스 커널 개발에 이용하려고 개발하였습니다. *( 역시 대단한 사람… )*

코드를 버전 관리함으로써, 배포후 major 버그를 발생시 빠르게 rollback을 하거나 수정된 코드만을 파악하여, 버그를 빠르게 찾거나, 한 프로젝트의 코드를 여러 사람이 함께 작업 할 수 있도록 도와주는 등 사용하면 많은 이점을 가지고 있다.

하지만 이 또한 도구일 뿐이다. 어떤사람이 도구를 어떻게 사용하느냐에 따라 최고의 도구가 될수도 부족한 도구가 될 수 있습니다. git 전략이 다양하게 존재하니 참고하여 사용하여 강력한 도구로 이용해 봅시다.



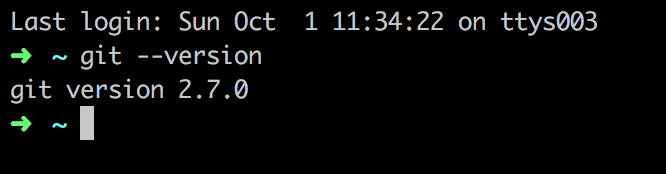
<http://nvie.com/posts/a-successful-git-branching-model/>

**Git 설치하기**

저는 mac 유저 이므로 mac 기준으로 설명하겠습니다. git 사이트에서 mac install package를 다운받은후 설치 할 수 있습니다.

[Git - Downloading Package  
You are downloading version 2.14.1 of Git for the Mac platform. This is the most recent maintained build for this…git-scm.com](https://git-scm.com/download/mac)

terminal에서 git 명령어를 입력하여 아래 그림같은 내용이 나온다면 설치가 완료된것입니다.



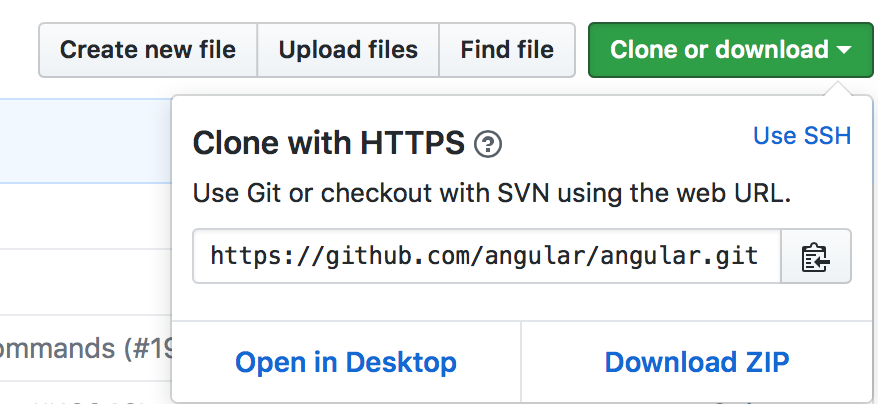
terminal

window에서도 git을 설치하면 git을 사용할 수 있는 terminal을 제공합니다.

**다른 개발자의 코드 받아오기**

코드를 git으로 다른사람들과 함께 작업하고 싶다면 git 서버가 필요합니다.

물론 git server를 직접 구축할 수 도 있지만, 이미 잘 만들어둔 서비스들이 존재합니다. 대표적은 예로 Github과 Bitbucket이 존재합니다. 이런 서비스를 통해 다른 사람들이게 코드를 공유할 수 있는데, 다른 사람의 코드를 받아오고 싶다면 어떻게 해야할까요?



github angular page

github 서비스에서는 download와 clone을 제공하고 있습니다. 저는 terminal을 사랑하기 때문에 terminal 위주로 설명하겠습니다. Github에서 angular를 받아오고 싶다면, 아래 명령어로 받아 올 수 있습니다.

*git clone https://github.com/angular/angular.git*

보통 GitHub에 등록된 opensouce의 경우 직접 clone을 하기 보다는 Github의 fork를 이용하여, 자신의 저장소에 복사한후 clone하여 작업하고 수정하는 경우가 많습니다. 이유인즉 opensouce에 직접 업로드할 권한이 없기때문에 Github의 자신의 저장소에 복사하면, 코드를 관리할 수 있게됩니다.

만약 opensouce에 버그를 발견하였고, 이를 수정하여 전달하고 싶다면, Github의 pull request를 활용해 수정사항을 전달 할 수 있고, 관리자가 허락해준다면 opensouce에 기여할 수 있게됩니다.

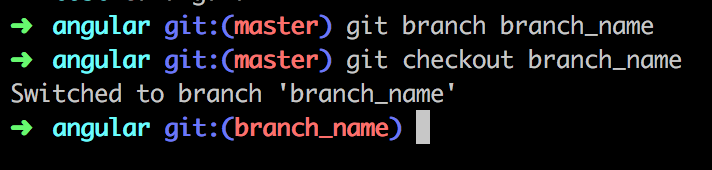
이미 다운 받았고, 최신 내용을 git 서버에서 받아와야 한다면, git pull과 git fetch로 받을수 있습니다.

*git pull // 코드를 받아와 변경점을 merge한다.  
git fetch // 코드를 받아온다.*

pull과 fetch의 가장 큰 차이는 merge를 하냐 하지 않느냐의 차이입니다. fetch의 경우 코드를 받아온후 즉시 코드를 반영하지 않아, 작업한 코드가 충돌이 일어날경우에 안전하지만, pull은 코드를 받아온후 즉시 merge하여 코드를 반영하기 때문에 충돌이 일어 날수 있습니다. 이 경우 충돌을 해결해주고 다시 commit 하면 문제 없습니다. 해당 내역은 아래에서 조금 더 설명하겠습니다.

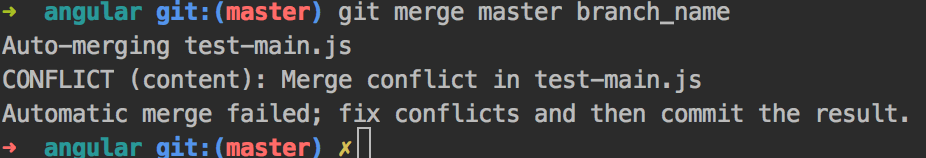
**Branch 관리하기**

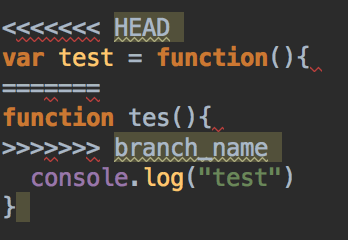
코드를 혼자만 사용한다면 꼭 branch를 활용할 필요가 없지만, 같이 관리해야한다면, branch를 나눠서 작업하는것은 좋은 선택입니다.



master에서 branch를 생성하면 master를 기준으로 branch가 생성됩니다. 코드들 기존 master 작업과 분리되며 언제든지 checkout을 통해 master와 branch를 이동하며 코드를 변경할 수 있게됩니다. 작업한 branch 내용을 master와 합쳐야한다면, merge 명령어로 브랜치를 합칠수 있습니다.

그런데 master에서 작업한 내용과 branch에서 작업한 내용이 같을때 merge하게되면 어떻게 될까요?

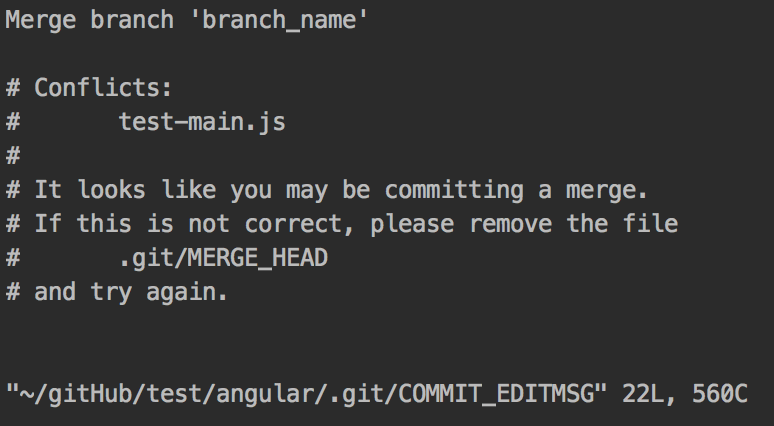




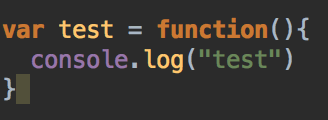
git merge master branch\_name

Git은 똑똑하게도 merge한 내용중 충돌하는 내용을 찾아서 수정해달라고 요청하게 됩니다. HEAD 영역부터 ===영역까지는 master에서 작업한 내용이고, ===부터 branch\_name 영역까지는 branch\_name이라는 브랜치에서 작업한 내용입니다.

HEAD 부터 branch\_name 까지의 영역의 내용을 모두 수정해주고, git add \* 으로 수정된 영역을 선택해주고 git commit으로 commit하면 충돌한 merge가 깨끗하게 합쳐집니다.



git commit



완료된 master 내용

**Commit 취소하기**

git은 코드의 상태를 uncommit / staying / commit 으로 나눠 볼수 있습니다.

1. 코드 파일의 내용을 바꾸면 uncommit이 됩니다.
2. 파일의 변경 내역 add 하면 index에 추가되고 staying이 됩니다.
3. 변경 내용을 commit으로 확정지을 수 있씁니다.
4. commit 이 완료되면 서버에 push할 수 있게 됩니다.

코드를 작성하다보면 잘못된 코드를 추가하거나 merge 할 수 있습니다. 이를 취소하고싶다면, git reset 으로 취소 할 수 있습니다.

*git reset // add 한 파일 취소  
git reset — -merge // merge 한 코드 취소*

만약 commit한 코드를 취소하고 이전코드로 돌아가고 싶다면, git reset HEAD^ 명령어를 이용하여 바로 직전에 commit한 코드로 되돌아 갈 있습니다.

하지만 이경우 작성하고 수정했던 코드는 모두 사라지게 됩니다. 만약 commit만 취소하고, 작성한 코드는 살리고 싶다면 어떻게 해야할까요? reset 명령어의 옵션중에 soft를 이용하면 됩니다.

*git reset — soft HEAD^ //commit 코드 살리기  
git reset — hard HEAD^ //commit하기 이전의 코드로 돌아가기*

git reset을 이용하여 작성한 코드를 초기화하고 서버로부터 다시 받아 올 수 도 있습니다.

*git reset — hard HEAD // HEAD를 기준으로 이전코드로 돌아갑니다.  
git pull // git 서버에서 코드를 다시 받아옵니다.*

**코드 잠깐 저장하고, 다른 브랜치로 이동하기**

A라는 Branch에서 작업을 하던중 급하게 들어온 요청 혹은 Bug가있어 B 브랜치로 이동해야할 일이 생겼습니다. 그런데 작업은 완료되지 않아서 commit 해두기에는 애매하고, 그렇다고 다시 작업하기에는 작업량이 있어 버릴수는 없는경우에 어떻게 해야할까요?

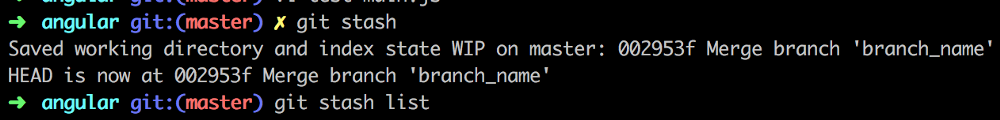
그럴 경우 잠깐 다른 공간에 변경 내역을 기록해 두고 브랜치를 변경해서 작업한후 다시 돌아와서 변경 내역을 불러와 계속 작업을 이어 나갈 수 있습니다.

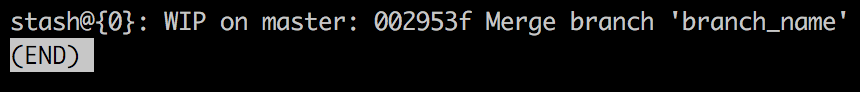
*git stash // 코드를 stash 영역에 잠시 저장하기*

stash 명령어를 이용하여 stash 영역에 코드를 잠깐 저장해두고 다시 돌아와 작업을 이어나갈수 있게 되었습니다. 해당 구조는 스택구조로 이루어져있기때문에 LIFO으로 작업을 불러올수 있습니다.

*git stash pop // 마지막에 저장한 코드 불러오기*

stash 영역이 기억이 나지 않는다면 git stash list로 목록도 확인해 볼수 있습니다.





git stash list

**느낀점**

제가 서비스를 개발하며, 자주 사용하던 기초적이고 꽤 자주 사용하던 git 명령어를 정리해 보았습니다. 저는 처음에 git을 사용할때는 정말 어렵게만 느껴지고, 어떻게 사용해야할지 막막하기만 하였는데, 이렇게 기초적인 명령어들이긴 하지만 꽤 많은 명령어를 쓸 수 있게 되었다니, 감개무량하기만 합니다. 상세하게 들어가면 더 많은 내용과 재미 있는 내용들이 있지만, 조금 더 준비해서 Lv100이 되면 도전해 봐야겠습니다.