













**환경 설정**  
git config --global --list   
현재 설정정보 조회할 수 있습니다. --global옵션은 전역설정에 대한 옵션이며 현재 프로젝트에만 적용할때는 주지 않습니다.  
git config --global user.name "사용자명"   
사용자명을 등록합니다 (필수)  
git config --global user.email "이메일주소"   
이메일 주소를 등록합니다. (필수)  
git config --global color.ui “auto”  
터미널에 표시되는 메시지에 칼라를 표시해줌  
  
  
git에 대해서 사전에 알아야 될 부분은 아래와 같습니다.

* git의 저장소는 3가지 단계로 나누어 집니다. 커밋한 소스가 보관되는 저장소와 현재 프로젝트 파일들이 있는 작업트리, 저장소와 작업트리사이의 버퍼영역으로 커밋될 대상이 저장되는 스테이징 영역입니다.
* git은 빈 디렉토리는 추적하지 않습니다.
* 형상관리를 하지 않을 파일은 [.gitignore 파일](http://www.kernel.org/pub/software/scm/git/docs/gitignore.html)에 추가합니다.
* HEAD는 현재 브랜치의 가장 최신커밋을 의미한다.
* 기본원격 저장소를 origin이라고 부릅니다.

**기본적인 명령어**  
git --version  
현재 git의 버전을 확인합니다.  
  
git init  
현재 디렉토리에 git 저장소를 생성합니다.  
  
git add 파일명  
git add는 2가지를 하는데 untracked files의 파일들을 git가 추적하도록 하거나 파일은 수정했지만 아직 스테이징 영역에 올라가지 않은(Changed but not updated) 파일들을 스테이징 영역에 올립니다. -i 옵션을 주면 대화형모드가 시작되며 파일의 일부분만 선택해서 스테이징하는 것이 가능합니다. -p 옵션을 사용하면 -i 대화형모드없이 바로 패치모드를 사용할 수 있습니다.  
  
git commit -m "커밋메시지"  
스테이징 영역에 올라가 있는 파일들을 커밋합니다. -m 은 커밋메시지를 주는 옵션으로 여러 줄의 커밋메시지를 쓸 경우 -m 을 여러개 사용할 수 있습니다. -a 옵션을 사용하면 스테이징에 올리는 작업과 커밋을 동시에 할 수 있습니다.(추적되지 않는 파일은 추가하지 않습니다.) -m을 사용하지 않을때 -v옵션을 사용하면 편집기에 커밋하려는 변경사항의 다른점을 보여줍니다. 특정파일만 커밋하려면 마지막에 파일명을 추가해주면 됩니다.  
  
git commit -C HEAD -a --amend  
지정한 커밋의 로그메시지를 다시 사용하여 기존커밋을 수정합니다. -c를 사용하면 기존메시지를 수정할 수 있는 편집기를 실행해 줍니다.  
  
git status  
커밋되지 않은 변경사항을 조회합니다.  
  
git diff  
스테이징영역과 현재 작업트리의 차이점을 보여줍니다. --cached 옵션을 추가하면 스테이징영역과 저장소의 차이점을 볼 수 있다. git diff HEAD를 입력하면 저장소, 스테이징영역, 작업트리의 차이점을 모두 볼 수 있다. 파라미터로 log와 동일하게 범위를 지정할 수 있으며 --stat를 추가하면 변경사항에 대한 통계를 볼 수 있습니다.  
  
git mv 파일명 새파일명  
기존에 존재하는 파일을 새파일로 이동합니다. 변경이력은 그대로 유지합니다.  
  
git checkout -- 파일명  
아직 스테이징이나 커밋을 하지 않은 파일의 변경내용을 취소하고 이전 커밋상태로 돌립니다. svn에서 revert와 동일합니다.  
  
  
  
  
**Branch와 Tag**  
git branch  
현재 존재하는 브랜치를 조회합니다. -r 옵션을 사용하면 원격저장소의 브랜치를 확인할 수 있습니다.   
  
git branch 브랜치명B 브랜치명A  
브랜치명A에서 새로운 브랜치 브랜치명B를 만듭니다. (git에서 기본 브랜치는 master라는 이름을 사용합니다.)  
  
git branch 브랜치명  
브랜치명의 새로운 브랜치를 만듭니다.(체크아웃은 하지 않습니다.)  
  
git branch -d 브랜치명  
브랜치를 삭제합니다.  
  
git branch -m 존재하는브랜치명 새로운브랜치명  
존재하는 브랜치를 새로운브랜치로 변경합니다. 이미 존재하는 브랜치명이 있을 경우에는 에러가 나는데 -M 옵션을 사용하면 이미 있는 브랜치의 경우에도 덮어씁니다.  
  
git tag 태그명 브랜치명  
브랜치명의 현재시점에 태그명으로 된 태그를 붙힙니다. git tag만 입력하면 현재 존재하는 태그 목록을 볼 수 있습니다.  
  
git checkout 브랜치명/태그명  
해당 브랜치나 태그로 작업트리를 변경합니다.   
  
git checkout -b 브랜치명B 브랜치명A  
브랜치명A에서 브랜치명B라는 새로운 브랜치를 만들면서 체크아웃을 합니다.  
  
git rebase 브랜치명  
브랜치명의 변경사항을 현재 브랜치에 적용합니다.  
  
git merge 브랜치명  
브랜치명의 브랜치를 현재 브랜치로 합칩니다. --squash 옵션을 주면 브랜치명의 모든 커밋을 하나의 커밋으로 만듭니다.  
  
git cherry-pick 커밋명  
커밋명의 특정 커밋만을 선택해서 현재 브랜치에 커밋으로 만듭니다. -n 옵션을 주면 작업트리에 합치지만 커밋은 하지 않기 때문에 여러개의 커밋을 합쳐서 커밋할 수 있습니다.  
  
  
  
  
**로그 관리**  
git log  
커밋로그들을 볼 수 있으면 -1나 -2같은 옵션을 주어 출력할 커밋로그의 갯수를 지정할 수 있습니다. --pretty=oneline 옵션을 주면 한줄로 간단히 보여주고 --pretty=format:"%h %s"처럼 형식을 정해줄 수 있습니다. -p 옵션을 사용하면 변경된 내용을 같이 보여줍니다. --since="5 hours" 이나 --before="5 hours"같은 옵션도 사용가능합니다. --graph 옵션을 주면 브랜치 트리를 볼 수 있습니다.  
  
git log 커밋명  
해당 커밋명의 로그를 볼 수 있습니다. 커밋명A..커밋명B (마침표2개)와 같이 입력하면 커밋명A이후부터 커밋명B까지의 로그를 볼 수 있습니다. ^은 -1과 동일해서 HEAD^라고 하면 최신바로 이전 커밋이고 HEAD^^^와 같이 쓸 수 있으며 HEAD~3을 하면 HEAD의 3개 이전의 커밋을 뜻합니다.  
  
git blame 파일명  
갈 줄 앞에 커밋명과 커밋한 사람등의 정보를 볼 수 있습니다.  
  
git blame -L 10,15 파일명  
-L 옵션을 사용하면 10줄부터 15줄로 범위를 지정해서 볼수 있고 15대신 +5와 같이 사용할 수 있습니다. 숫자의 범위 대신 정규식도 사용이 가능합니다.  
  
git blame -M 파일명  
-M 옵션을 사용하면 반복되는 패턴을 찾아서 복사하거나 이동된 내용을 찾아줍니다.  -C -C 옵션을 사용하면 파일간의 복사한 경우를 찾아줍니다. -C -C는 git log에서도 사용가능하며 내용의 복사를 찾을때는 git log에서 -p옵션을 사용합니다.  
  
git revert 커밋명  
기존의 커밋에서 변경한 내용을 취소해서 새로운 커밋을 만듭니다. -n옵션을 사용하면 바로 커밋하지 않기 때문에 revert를 여러번한 다음에 커밋할 수 있습니다.(항상 최신의 커밋부터 revert해야 합니다.)  
  
git reset 커밋명  
이전 커밋을 수정하기 위해서 사용합니다. --soft 옵션을 사용하면 이전 커밋을 스테이징하고 커밋은 하지 않으며 --hard옵션은 저장소와 작업트리에서 커밋을 제거합니다. git reset HEAD^와 같이 입력하면 최근 1개의 커밋을 취소할 수 있습니다.  
  
git rebase -i 커밋범위  
-i옵션으로 대화형모드로 커밋 순서를 변경하거나 합치는 등의 작업을 할 수 있습니다.  
  
  
  
  
**원격저장소**  
git clone 저장소주소 폴더명  
원격저장소를 복제하여 저장소를 생성합니다. 폴더명을 생략가능합니다.  
  
git fetch  
원격저장소의 변경사항 가져와서 원격브랜치를 갱신합니다.  
   
git pull  
git fetch에서 하는 원격저장소의 변경사항을 가져와서 지역브랙치에 합치는 작업을 한꺼번에 합니다. 파라미터로 풀링할 원격저장소와 반영할 지역브랜치를 줄 수 있습니다.  
  
git push  
파라미터를 주지 않으면 origin 저장소에 푸싱하며 현재 지역브랜치와 같은 이름의 브랜치에 푸싱합니다. --dry-run 옵션을 사용하면 푸싱된 변경사항을 확인할 수 있습니다. 로컬에서 tag를 달았을 경우에 기본적으로 푸싱하지 않기 때문에 git push origin 태그명이나 모든 태그를 올리기 위해서 git push origin --tags를 사용해야 합니다.  
  
git remote add 이름 저장소주소  
새로운 원격 저장소를 추가합니다.  
  
git remote  
추가한 원격저장소의 목록을 확인할 수 있습니다.  
  
git remote show 이름  
해당 원격저장소의 정보를 볼 수 있습니다.  
  
git remote rm 이름  
원격저장소를 제거합니다.  
  
  
  
  
**서브모듈**  
git submodule  
연관된 하위모듈을 확인할 수 있습니다.  
  
git submodule add 저장소주소 서브모듈경로  
새로운 하위모듈을 해당경로에 추가합니다. 추가만하고 초기화 하지는 않으며 커밋해쉬앞에 마이나스(-) 표시가 나타납니다.  
  
git submodule init 서브모듈경로  
서브모듈을 초기화 합니다.  
  
git submodule update 서브모듈경로  
서브모듈의 변경사항을 적용합니다.(저장소의 최신커밋을 추적하지 않습니다.)  
  
  
  
  
**기타 명령어**  
git archive --format=tar --prefix=폴더명/ 브랜치혹은태그 | gzip > 파일명.tar.gz  
git archive --format=zip --prefix=폴더명/ 브랜치혹은태그 > 파일명.zip  
해당 브랜치나 태그를 압축파일로 만듭니다. --prefix를 주면 압축하일이 해당폴더 안에 생성되도록 할 수 있습니다.  
  
git mergetool  
설정에 merge.tool의 값에 있는 머지툴을 찾아서 실행합니다.  
  
git gc  
저장소의 로그를 최적화 합니다. 로그가 변경되지는 않고 저장하는 방식만 최적화 합니다. --aggressive 옵션을 주면 더 자세하게 최적화합니다.  
  
git rev-parse --show-toplevel  
git 저장소내에서 입력하면 루트디렉토리를 알려줍니다.