


- git rm
- git mv
- git clean
- 5. Branch와 Merge
  - git branch
  - git merge
- 6. log
  - git log
- 7. 공유 & 업데이트
  - git fetch
  - git pull
  - git push
  - git remote
- 8. 임시 저장
  - git stash
- 9. 마치며
  - 참고



강남역 1번출구

연세우노비뇨기과의원 강남  
강남역 1번 출구, 비뇨기과전문의

대한의사협회 의료광고심의필 제 191022-증-100487호

## 1. 시작하며

자주 쓰는 git 명령어를 git commands cheat sheet 를 참고하여 정리해보자.

## 자주 쓰는 Git Command

# 현재 directory 의 모든 파일을 Staging Area 로 이동

```
git add .
```

# file 들의 tracking 상태 보기

```
git status
```

# Staging 의 파일들 commit 하기

```
git commit -m "message"
```

# 저장소에 commit 반영하기

```
git push
```

# 저장소에서 commit 가지고 오기

```
git pull
```

# remote origin의 development branch merge

```
git merge origin/development
```

# 한 줄로 그래프 형태로 commit 히스토리 보기

```
git log --oneline --graph
```

# remote에서 삭제된 brach를 local 에서도 깔끔하게 삭제

```
git fetch origin --prune
```

## 2. Setup 초기 설정

### 사용자 이름, Email 설정

```
✓ git config --global user.name "[firstname lastname]"
```

출력되는 command line를 읽기 쉽도록 자동으로 색깔 설정한다.

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

## 사용자 설정 정보 확인

전체 설정 정보 확인을 위해 아래 명령어를 실행하거나 혹은 .gitconfig 파일을 확인한다.

```
git config --list
```

사용자 정보는 아래와 같이 확인한다.

```
git config --global user.email  
git config --global user.name
```

## 3. Git 프로젝트 생성하기, 가져오기

아래 두 가지 방법 중 한가지로 git 저장소를 시작할 수 있다.

- 아직 버전관리를 하지 않는 로컬 디렉토리 하나를 선택해서 Git 저장소를 적용하는 방법
- 다른 어딘가에서 Git 저장소를 Clone 하는 방법

### 1) git init: 기존 디렉토리를 Git 저장소로 만들기

프로젝트 directory로 가서 git init을 실행하면 새로운 Git 저장소가 만들어진다.

```
# 프로젝트 directory로 이동  
cd [my_project path]  
  
# README 파일 생성  
echo "# my_project" >> README.md
```

```
# 관리할 file을 Staging Area 추가
```

```
git add README.md
```

```
# Staging Area 에 추가된 파일 commit
```

```
git commit -m "first commit"
```

```
# 최초 등록된 master branch 대신 main branch 사용하도록 변경
```

```
git branch -M main
```

```
# 원격 remote repository에 추가
```

```
git remote add origin [git URL]
```

```
# git push
```

```
git push -u origin main
```

## 2) git clone: git repository 가져오기

기존에 사용 중인 저장소를 clone 해서 가져올 수 있다.

```
git clone [url]
```

## 4. Snapshot 스냅샷 다루기

Git 스냅샷 즉 특정 상태의 버저닝을 생성한다. Staging Area에 파일을 추가하고 commit, push 정도만 알면 된다.

```
git add
```

working directory 에서 파일/폴더를 Staging Area(“index”)에 등록하여 git 으로 관리를 시작한다.

```
git add [file]
```



git의 현재 상태 확인한다. Staging Area, UnStage Area 에 있는 파일 및 Untracked 상태의 파일을 확인 할 수 있다.

```
git status
```

## git diff

git diff 명령은 두 트리 개체의 차이를 보고 싶을 때 사용한다.

```
# stage 되지 않은 변경 비교
git diff
```

```
# stage 되어 있으나 아직 commit 되지 않은 변경 비교
git diff --staged
```

## git commit

git commit 명령은 Staging Area의 모든 파일을 커밋한다. 저장소에는 하나의 스냅샷으로 기록된다. 그 뒤 현재 Branch가 새 커밋을 가리키게 한다.

```
git commit -m "[커밋 상세 메시지]"
```

--amend 옵션으로 최근 커밋을 재작성할 수 있다

```
git commit --amend
```

아래와 같이 commit에 깜박한 파일이 있으면 아래와 같이 수정 가능하다. 이렇게 --amend 옵션으로 커밋을 고치는 작업은 이전의 커밋을 완전히 새로 고쳐서 새 커밋으로 변경하는 것을 의미한다. 이전의 커밋은 일어나지 않은 일이 되는 것이고 당연히 히스토리에도 남지 않는다.

```
git commit -m 'initial commit'
```

## git reset

git reset 명령은 되돌리는(Undo) 명령이다. git reset 명령은 매우 위험하므로 주의하여 사용하여야 한다.

```
# Staged 상태의 파일을 Unstage 로 변경
git reset
```

```
# staging area가 초기화되고 working tree를 특정 커밋시점으로 덮어쓰기
git reset --hard [commit]
```

```
# 이전 Head 시점으로 초기화. working directory, 최근 commit 이력이 삭제된다.
git reset --hard HEAD^
```

## git rm

Staging Area나 Working directory에 있는 파일을 삭제한다.

```
git rm [file]
```

## git mv

파일/폴더의 이름을 변경한다.

```
git mv [existing-path] [new-path]
```

mv 명령어는 사실 아래와 같이 동작한다. 즉 이름이 바뀐 파일을 삭제하고 다시 add 하는 것과 똑같다.

```
mv README.md README
```

```
git rm README.md
```

```
git add README
```

working directory에서 필요 없는 파일을 삭제한다. 충돌로 생긴 파일이나 build artifact 파일을 삭제할 때 편리하다.

```
git clean
```

## 5. Branch와 Merge

### git branch

branch를 관리하는 명령어 이다.

```
# branch 목록 조회, *가 현재 branch  
git branch
```

```
# 원격 branch 목록 조회  
git branch -r
```

```
# 전체 branch 목록 조회  
git branch -a
```

branch 를 신규 생성한다.

```
# branch 생성  
git branch [branch-name]
```

Branch를 변경하고 해당 파일을 Working directory로 복사한다.

```
# branch checkout  
git checkout
```



git merge

# 특정 branch를 현재 checkout 된 branch에 merge

```
git merge [branch]
```

# remote origin의 main branch merge

```
git merge origin/main
```

# local 저장소의 main branch merge

```
git merge main
```

## 6. log

### git log

commit 히스토리를 시간의 역순으로 보여준다.

# 커밋 히스토리를 시간 순으로 조회

```
git log
```

# 최근 number개만 노출

```
git log -[number]
```

# oneline 으로 히스토리 노출

```
git log --oneline
```

# graph 로 출력

```
git log --graph
```

## 7. 공유 & 업데이트

### git fetch



--prune 옵션으로 remote 저장소에 지워진 브랜치를 local 반영하여 local의 불필요한 branch를 삭제한다.

```
git fetch --prune
```

## git pull

remote의 commit을 가져오고 병합한다. git fetch 와 git merge 명령을 순서대로 실행하는 것과 같다.

```
git pull
```

## git push

로컬 저장소의 commit 내역을 remote 저장소로 전송한다.

-u 옵션은 upstream repository를 설정해준다. 즉 한번 설정한 후로는 git push, git pull 만 간단히 쓸 수 있다.

```
git push [remote] [branch]
```

```
git push -u origin main
```

## git remote

원격 저장소 설정 관리 도구다.

```
# 현재 프로젝트에 등록된 리모트 저장소를 확인
```

```
git remote
```

```
# 단축이름과 URL을 함께 조회
```

```
git remote -v
```

✓ # git url 에 remote(저장소) 이름으로 등록한다.

## 8. 임시 저장

### git stash

현재 작업을 임시 저장하는데 사용한다.

A라는 작업을 진행하다가 잠깐 다른 B 작업을 우선 할 일이 생겼다 이 때 A를 커밋하지 않고 Stash 했다가 B를 끝낸 후 다시 A라는 작업을 진행 할 수 있다. Branch에 상관없이 아직 끝내지 않은 수정사항을 스택에 잠시 임시 저장했다가 나중에 다시 적용할 수 있다.

```
# modified, staged 변경 내용 임시 저장
git stash
```

```
# stashed 조회
git stash list
```

```
# stash 내역 working directory로 추가
git stash pop
```

```
# stash file 제거
git stash drop
```

## 9. 마치며

새로운 프로젝트를 시작할 때면 항상 나를 찾는 사람 에게 도움이 되길 바라며 🙏

### 참고

<https://education.github.com/git-cheat-sheet-education.pdf>

<https://git-scm.com/book/ko/v2>

<http://ohyecloudy.com/ddiary/2017/06/20/til-git-prune-with-fetch-or-pull/>

