Configuration

git config 확인하기

```
$ git config -l
$ git config --global -l
```

git config 수정하기

```
$ git config --global --edit
```

사용자명/이메일 설정하기

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

file permission 무시하기

```
$ git config
  core.fileMode false
```

Login credentials 캐쉬 설정

```
$ git config --global credential.helper cache
```

Login credentials 캐쉬 설정

```
$ git config --global credential.helper cache
```

Repository

repository 생성

```
$ git init
$ git init PROJECT_NAME
```

staging area 로 파일 추가

```
# 변경된 전체 파일 추가
$ git add .

# 변경된 1개 파일 추가
$ git add FILE-NAME

# PATH 내의 변경된 파일 추가
$ git add FILE-PATH
```

브랜치 상태확인

```
$ git status
```

commit 하기

```
# editor 로 commit message 작성
$ git commit
# command line 으로 commit message 작성
$ git commit -m "COMMIT-MESSAGE"

# tracked files 을 add 하고 commit 을 동시에 한다
$ git commit -a -m "COMMIT-MESSAGE"

# 자체서명 commit 을 만든다
$ git commit -as
$ git commit -as -m "commit message"
```

Log

commit Log 확인

```
# commit history 를 본다
$ git log

# commit의 변경 내용까지 확인
$ git log -p

# NUMBER 갯수 만큼 log 를 보여줌
$ git log -NUMBER

# log 를 한줄로 보여줌
$ git log --oneline

# log 를 커밋해시와 저자만 보여줌
$ git log --pretty=format:"%H, %an"

# merge 된 브랜치 그래프를 보여줌
$ git log --oneline --decorate --graph --all

# first hash 부터 last hash 까지 commit log 를 보여줌
$ git log <first_commit_hash>~..<last_commit_hash>

# commit hash 의 log 만 보여줌
$ git log <commit_hash>~..<commit_hash>

# commit hash 의 commit 내용을 보여줌
$ git show <commit_hash>

# log stats 를 조회한다
$ git log --stat
```

commit 전에 변경내역을 확인

```
# unstaged changes 와 비교한다
# add 로 추가되지 않은 변경된 소스코드를 보여준다
# working directory 와 Index 를 비교
$ git diff

$ git diff filename

# staged changes 와 비교한다
# add 로 추가된 소스코드를 커밋로그 첫번째(HEAD)와 비교해준다
# index 와 HEAD 를 비교
$ git diff --staged
$ git diff --cached

# working directory 와 HEAD 를 비교
$ git diff HEAD

# working directory 와 HEAD^ 를 비교
$ git diff HEAD^
```

tracked files 에서 삭제

```
$ git rm filename
```

file 이름 변경

```
$ git mv oldfile newfile
```

file 원복

```
# revert unstaged changes
$ git checkout filename

# revert staged changes
$ git reset HEAD filename
$ git reset HEAD -p
```

최근 commit 수정

```
# overhead commit 을 변경한다
# commit hash 가 바뀜
$ git commit --amend -a
# HEAD 의 commit 메시지만 변경 한다
$ git commit --amend -m "CHANGE-COMMIT-MESSAGE"
```

브랜치 이름 변경하기

```
$ git branch -m OLD-BRANCH-NAME NEW-BRANCH-NAME
```

마지막 commit 으로 원복

```
# 가장 최근 commit 으로 원복
$ git revert HEAD

# 되돌릴 COMMIT-ID 로 Revert 한다
$ git revert COMMIT-ID
# 또는
# HEAD~3 최근 3개 전의 commit 으로 Revert 한다
$ git revert --no-commit HEAD~3
$ git commit -m "Revert Comit A,B,C"
```

Branch

새로운 브랜치 만들기

```
# NEW-BRANCH-NAME
$ git branch NEW-BRANCH-NAME

# 브랜치로 전환
$ git checkout NEW-BRANCH-NAME

#-b 옵션은 현재 branch 를 복사해서 NEW-BRANCH-NAME 으로 만들고 switch
$ git checkout -b NEW-BRANCH-NAME
```

브랜치 리스트 조회

```
$ git branch
$ git branch --list
```

브랜치 삭제

```
$ git branch -d BRANCH-NAME
# 강제로 브랜치 지우기
$ git branch -D BRANCH-NAME
```

브랜치 병합

```
#현재 브랜치에 BRANCH-NAME 을 병합한다
$ git merge BRANCH-NAME
```

병합된 브랜치 리스트 조회

```
# 현재 브랜치에 병합된 브랜치 리스트를 보여준다
$ git branch -a --merged
```

브랜치 병합을 원복

```
$ git merge --abort
```

원격 Repoistory

github 의 remote repository 연결하기

```
# remote repository 를 조회한다
$ git remote
# remote repository 의 주소도 조회한다
$ git remote -v

# origin 이라는 명칭으로 github 의 repository 가 정의됩니다
$ git remote add origin [깃허브 URL]

$ git remote show origin
```

remote repository 를 복사해오기

```
$ git clone
```

github repository 로 소스 올리기

```
$ git push -u REMOTE-NAME BRANCH-NAME

$ git push -u origin master
# origin 은 git remote add origin 에서 설정한 명칭 입니다

# 혼자 쓰는 거면 이렇게 git push 만 쳐도 됨
$ git push
# git push -u origin master 와 동일하게 처리됨
```

원격 브랜치 확인

```
$ git branch -r
```

원격 repository 와 local repository 병합하기

```
# github repository 를 가져와서
$ git fetch REMOTE-NAME
# 로컬 repository 와 병합한다
$ git merge REMOTE-NAME/BRANCH-NAME
```

병합 없이 remote branch 를 가져오기

```
$ git remote update
```

원격 브랜치 삭제

```
$ git push --delete REMOTE-NAME :BRANCH-NAME
```

원격 브랜치 이름 변경하기

```
git push origin :OLD-BRANCH-NAME NEW-BRANCH-NAME
```

MISC

commit history 변경

```
$ git rebase BRANCH-NAME
$ git rebase -i BRANCH-NAME
```

원격 브랜치의 최신정보 가져오기

```
# pull = fetch + merge
# git pull origin master
$ git pull REMOTE-NAME BRANCH-NAME

# 원격 저장소에서 다운만 한다 (merge 는 따로 해야 한다)
$ git fetch REMOTE-NAME
# 가져온 정보를 로컬브랜치와 비교 하고 직접 병합 해준다
# git diff HEAD origin/master
# git log --decorate --all --oneline
# git merge origin/master

# 모든 리모트 정보를 업데이트 한다
# fetch 수행됨
$ git remote update
```

prune

```
# 새로 추가되었거나 삭제된 리모트 브랜치의 정보를 최신화 한다
$ git remote prune REMOTE-NAME

# 리모트 저장소에서 삭제된 브랜치를 로컬 저장소에도 적용
$ git pull --prune
$ git fetch --prune

$ git prune

# 옵션으로 적용도 가능하다
$ git config --global fetch.prune true
```

Stash/un-stash files

변경저장

```
$ git stash

# stash list 를 본다
$ git stash list
```

변경저장 해제

```
$ git stash pop
```

브랜치 변경을 동기화

변경사항 확인

```
$ git diff --cached

# unstage 로 원복
$ git reset <file_path>

# 변경사항에 문제 없으면 commit 한다
$ git commit
```

untracked 파일 또는 폴더 제거하기

```
# To remove untracked files
$ git clean -f
# TO remove untracked directories
$ git clean -fd
```

마지막 commit 으로 reset 하기

```
$ git reset hard origin/BRANCH-NAME
```

모든 변경사항 취소하기

```
$ git reset --hard
```

Github

Repository 안의 파일 검색 : 단축키 T

Repository에서 파일을 찾을 때 단축키 T를 누르면 빠르고 쉽게 파일을 검색할 수 있다.

전체 Repository에서 찾기 : 단축키 /

소스 수정을 하지 않고 단순히 검색만 하려면 단축키 / 를 통해 빠르게 검색할 수 있다. 이 검색 기능은 계정 내 전체 Repository에서 소스 코드 뿐 만 아니라 Commit 메세지, Issue, Wiki 등의 내용도 찾아준다.

Git Repository URL 단축하기 : http://git.io

http://git.io/에서는 GitHub Repository에 대해 https://git.io/repo 형태로 URL을 단축해주는 기능을 제공하고 있다.

소스 수정한 사람 찾기 : 단축키 B

Git에는 Git Blame 이라는 명령어가 있다. 누가 소스를 수정했는지 검색할 수 있는 기능이다. GitHub에서는 소스 파일에서 단축키 B를 누르면 된다.

Pull Request (PR) 되돌리기

GitHub에서 Pull Request를 잘못 보냈을 때 이를 되돌릴 수 있다. 의외로 이 기능을 잘 모르는 사람도 있어 적어본다. PR 리스트에서 되돌리고 싶은 PR을 선택하고 Revert 버튼을 누르면 이전 PR이 Undo되며 새로운 PR을 보낼 수 있다.