**참고 사이트**

<https://evan-moon.github.io/2019/07/25/git-tutorial/>

<https://codingcoding.tistory.com/1080>

<https://tagilog.tistory.com/377>

**Git 용어 해설**

Repository: 파일 등의 데이터 저장소

local: 내 PC의 저장 장소

remote: GitHub의 저장 장소 (클라우드 비슷한 개념)

Branch: 개별적으로 작업하는 장소. 별도의 작업 디렉토리

Clone: 저장소를 통째로 복제함

Pull: 원격 저장소의 merge 된 내용을 로컬 저장소에 반영

Staged: 현재 수정한 파일을 곧 commit할 것이라고 표시한 상태

Commit: 편집한 파일을 local 저장소로 보냄

Push: local 저장소에서 remote 저장소로 전송

Origin: 따로 지정하지 않았을 때 지정되는 remote 저장소 name

Main(구 master): default branch name. 따로 지정하지 않으면 local에도 main(구버전이면 master)으로 생김

**Git 일반적인 사용 순서**

git clone (혹은 git init + git remote add + git pull) -> (branch 생성) -> 파일 수정 및 추가 -> git add -> git commit -> git push

현재(20210707) git\_pull.bash 와 git\_push.bash 스크립트를 사용 중

처음에 git\_pull.bash를 통해 파일들을 가져온 후 변경 사항들을 git\_push.bash {filename} 방식으로 업로드

**Git (자주 쓰는) 명령어**

git clone {URL}: URL에 위치한 repository를 local 저장소로 가져옴

-b {branchname} {URL}: URL에 위치한 특정 branch를 clone

git remote: 어떤 remote 저장소가 지정되어 있는지 확인

add {remote name} {URL}: URL 주소를 원하는 name의 remote 저장소로 추가

rm {remote name}: 지정된 remote 저장소 제거

-v: 지정된 remote 저장소의 상세 주소 확인 가능

git pull {remote name} {branchname}: remote 저장소의 특정 branch를 가져와서 merge

git branch: 현재 branch 목록 (현재 branch 위치는 \*로 표시)

-a: 전체 branch

-d {branchname}: branch 삭제

--merge: merge가 완료된 branch 목록

git checkout {branchname}: 해당 branch로 이동

-b: branch를 새로 만드는 동시에 이동

git merge {branchname}: 현재 branch에 {branchname}의 branch를 merge

git add {file or directory}: 해당 file이나 directory의 변경 사항을 stage에 올림 (git add . 을 사용할 경우 현재 위치에 있는 모든 file을 지정)

git status: stage에 담긴 변경 사항들을 확인

-v: 어떤 파일의 어떤 부분이 변경되었는지도 확인 가능

git commit: git add 했던 내용을 반영 (-m 옵션 해제시 수동으로 입력하라고 알려줌)

-m {message}: 원하는 message를 추가

git push {remote name} {branchname}: remote 저장소에 특정 branch를 push

**Confused concept**

clone vs pull

git clone: local에 아무것도 없는 상태에서 remote 저장소의 데이터를 가져옴

git pull: local에 이미 있고, remote 저장소의 수정 상태를 반영하기 위해 하는 것

엄밀하게는 아니지만 대략적으로는 clone = git\_init + git remote add + git pull

fetch vs pull

git fetch: remote 저장소의 commit된 내용들을 local 저장소로 가져옴. merge 필요

git pull: remote 저장소의 정보들을 가져오면서 자동으로 local branch에 merge

엄밀하게는 아니지만 대략적으로는 fetch + merge = pull

**Error 목록**

git\_pull이나 git\_push 후 다음과 같은 error

error: The requested URL returned error: 403 Forbidden while accessing

해당 주소에 접근 권한이 없을 때 생김(보통 private repository인 경우)  
URL을 다음과 같이 변경함으로 해결 가능

git remote set-url {remote name} https://{username}@github.com/{username}/

{repositoryname}.git

git\_push 후 다음과 같은 error

error: failed to push some refs to https://{username}@github.com/{username}/

{repositoryname}.git '

To prevent you from losing history, non-fast-forward updates were rejected

Merge the remote changes before pushing again. See the 'Note about

fast-forwards' section of 'git push --help' for details.

remote 저장소의 내용이 내가 pull 한 이후 변경되었을 경우에 생김

git pull을 다시 해온 후 push 해야함 (pull 하는 과정에서 추가된 파일은 상관 없지만 똑같은 파일명이 있을 경우 변경사항이 사라지지 않도록 주의)

**Script**

git\_pull.bash (처음 한 번만 실행)

|  |
| --- |
| #!/bin/bash |
|  |  |
|  | repo='Source' # Remote repository name you want to pull |
|  | reponame='origin' # Bookmark for the remote repository |
|  | branch='main' # Branch name you want to pull |
|  |  |
|  | rm -rf ./.git |
|  |  |
|  | git init |
|  | git config --global user.name "Jung-ocean" |
|  | git config --global user.email j.jung@snu.ac.kr |
|  | git remote add ${reponame} https://github.com/Jung-ocean/${repo}.git |
|  | git remote set-url ${reponame} https://Jung-ocean@github.com/Jung-ocean/${repo}.git |
|  | git pull ${reponame} ${branch} |

git\_push.bash {filename}

|  |
| --- |
| #!/bin/bash |
|  |  |
|  | # You have to run git\_pull.bash first |
|  | # Changing or adding files should be done after git\_pull.bash |
|  | # Run git\_push.bash after Changing or adding files |
|  |  |
|  | reponame='origin' # Bookmark for the remote repository. This should be the same as the name determined by git\_pull.bash |
|  | branch='main' # Branch name you want to push |
|  | file2upload=$1 # File or direcotry you want to push |
|  |  |
|  | git add ${file2upload} |
|  | time=`date "+%Y%m%d %H:%M:%S"` |
|  | git commit -m "${time}" |
|  | git checkout -b ${branch} |
|  |  |
|  | git push -u ${reponame} ${branch} |

**git에 대한 용엽이형 정리 내용 (20210705)**

시간순으로 프로젝트의 스냅샷을 저장

거의 모든 명령을 로컬에서 실행 (네트워크에 접속하고 있지 않아도 커밋 가능)

Git으로 무얼 하든 Git 데이터베이스에 데이터가 추가됨. 되돌리거나 데이터를 삭제할 방법은 없음. 커밋하고나면 데이터를 잃어버리기 어려움

Modified -> Staged -> Committed -> push

Modifed : 수정만하고 로컬 데이터베이스에 커밋하지 않은 것

Staged : 커밋할 것이라고 표시한 것

Committed : 로컬 데이터메이스에 안전하게 저장됨

Push : 리모트 데이터베이스에 커밋된 파일 올림

origin : remote repository의 별칭

git config로 설정 내용을 확인하고 변경할 수 있음

~/.gitconfig, ~/.config/git/config : 현재사용자에게만 적용되는 설정. git config --global 옵션으로 이 파일을 읽고 쓸 수 있음. 특정 사용자의 모든 저장소 설정에 적용

.git/config : .Git 디렉토리에 있고, 특정 저장소(현재 작업 중인 프로젝트) 에만 적용. --local 옵션 사용시 이 파일 사용하도록 지정 (기본)

설정은 로컬일수록 우선

git config --global user.name “Yong-Yub KIM”

git config --global user.email kimyy308@snu.ac.kr

git config --global init.defaultBranch main (로컬 브랜치는 master, 웹은 main이라 달라서 불편한데 한번 main으로 바꾸면 신경쓰지 않아도 됨)

git config --global alias.ci commit : git ci 만으로도 commit할 수 있게 alias

git config --global alias.unstage 'reset HEAD --' : unstage로 변경하는 명령을 alias

git config --global alias.last 'log -1 HEAD' : 최근 커밋 쉽게 확인

git config --list 로 설정 내용 확인 가능

git help 로 도움말. git help config 라면 config에 대한 도움말

git init 프로젝트의 디렉토리로 이동 후, 실행하면 .git이라는 하위 디렉토리를 만듦

git clone https://github.com/kimyy308/SSH\_KHOA.git : 웹상의 저장소를 Clone 하는 것. user@server:path/to/repo.git 와 같이 SSH 프로토콜을 사용할 수도 있음

git status : 현재 git 상태 확인. tracked -> 커밋 명령을 실행하면 커밋될 파일들, untracked -> 커밋 명령 실행해도 커밋되지 않는 파일들

git add 파일이름 : modified 상태의 파일을 stage에 올림 (커밋 전 단계)

.gitignore : 이 파일안에 적힌 파일들은 git이 관리하지 않음. 자세한 설명은 책 참고

git commit : 커밋. 첫 줄에 적고싶은 메시지를 적고 저장해야 커밋이 됨

git commit -m ‘적고싶은 메시지’ : 인라인 메시지와 함께 커밋

git commit -a : modified 파일들이 자동으로 stage에 추가된 후 commit 됨

git commit --amend : 기존의 커밋을 재작성 (덮어쓰기)

사용 예시 (파일 하나 빠뜨렸을 때, 기존의 잘못된 커밋을 새로운 커밋으로 덮어쓰기. 한 개의 커밋으로 기록됨)

$ git commit -m 'initial commit'

$ git add forgotten\_file

$ git commit --amend

git rm : tracked 상태의 파일을 삭제. --cached 옵션을 주면 파일은 유지, -f 옵션을 주면 워킹 디렉터리 내 파일까지 삭제 ex ) git rm log/\\*.log (와일드카드 사용 시 \ 붙여야)

git mv : 파일 이름 변경 (mv -> git rm -> git add)

git log : 커밋 히스토리 조회. 가장 최근의 커밋이 가장 먼저 나옴. 커밋의 SHA-1 체크섬, 저자 이름, 저자 이메일, 커밋 날짜, 메시지를 보여줌.

-p 옵션을 걸면 각 커밋의 diff 결과를 보여줌

-2 옵션을 걸면 최근 두 개의 결과만 조회

--pretty=format 포맷맞춘 결과만 조회

--since=2.weeks 최근 2주 내 커밋만 조회

이외에도 옵션 매우 많음

git diff : unstaged 상태인 변경 부분들 비교. git diff --cached 나 --staged는 staged 상태인 파일 변경부분 비교

git reset HEAD <파일이름> : staged 파일을 unstaged로 변경

git checkout -- <파일이름> : 수정했던 파일을 워킹 디렉토리에 처음 checkout한 내용으로 되돌림

git remote : 현재 프로젝트에 등록된 리모트 저장소 확인. 저장소를 clone하면 origin 이라는 리모트 저장소가 자동으로 등록됨. -v 단축이름과 URL을 함께 볼 수 있음

git remote add <단축이름> <url> : 기존 워킹디렉토리에 새 리모트 저장소를 쉽게 추가 가능. 한번 추가하면 url 대신 단축이름을 사용할 수 있음

git fetch <remote> : 로컬에는 없지만, 리모트 저장소에는 있는 데이터를 모두 가져옴. 그러면 리모트 저장소의 모든 브랜치를 로컬에서 접근할 수 있음. 데이터를 가져오긴 하나 자동으로 merge는 안하므로, 수동으로 해야 함

git pull <remote> <branch> : 리모트 저장소 브랜치에서 데이터를 가져올 뿐만 아니라 자동으로 로컬 브랜치와 Merge 시킬 수 있음. 가장 중요. 작업 시작 전에 remote 서버와 동기화 시켜줘야 함

git push <remote> <branch> : upstream 저장소에 push. 리모트 저장소에 쓰기 권한이 있고, clone하고 난 이후 아무도 upstream 저장소에 push하지 않았을 때만 사용할 수 있음. 다른 사람이 push 한 후에 push 하려면 안됨. 먼저 다른 사람이 작업한 것을 가져와서 merge한 후에 push 가능

git remote show <remote> : 리모트 저장소의 구체적인 정보 확인

git remote rename <단축이름> <바꿀이름> : 리모트 저장소 이름 변경

git remote rm <remote> : 리모트 저장소 삭제

git -a v1.4 -m “메시지” : annotated 태그 작성

git show : 태그 정보 및 커밋 정보 모두 확인

예전 커밋에 대해서도 태그 가능. git log --pretty=oneline 후,

git tag -a v1.2 9fceb02 이런식으로 명령의 끝에 체크섬 명시해서 태그하면 됨

git push <branch> <태그 이름> 태그를 리모트 서버에 전송

git branch <branch이름> : 브랜치 만들기

git branch -d <branch이름> : 브랜치 삭제

git checkout <branch이름> : 다른 브랜치로 이동

예시 :

git checkout testing 으로 testing 브랜치로 이동 후

git commit -a -m ‘made a change’ 로 커밋

git checkout master 로 다시 기존 브랜치로 돌아옴

이렇게 되면 임시로 수정한 파일을 커밋하고, 이전에 작업하던 시점으로 워킹디렉토리 파일들을 되돌려서 다시 작업을 진행할 수 있음. 이후 master로 다시 커밋을 하면, 분리되어 히스토리가 진행되고 이후에 Merge를 통해 branch를 합칠 수 있음

git log --oneline --decorate --graph --all 와 같은 형태로 브랜치와 커밋의 히스토리를 깔끔하게 볼 수 있음

git checkout <master> 후 git merge <hotfix> : hotfix가 최근 branch인데, master를 hotfix가 가리키는 커밋으로 이동시키는 방식으로 merge. 이후 git branch -d hotfix로 master만 남기면 hotfix는 없애고, master를 갱신할 수 있음.

만약 서로 다른 길을 가고 있던 branch들을 merge 시킬 경우, 별도의 커밋(Merge 커밋)이 생기며, 현재 checkout 중인 branch가 Merge 커밋을 가리키도록 이동한다. 만약 다른 분기가 같은 부분을 수정하면 git은 merge하지 못하고 conflict message를 출력한다. git status로 자세한 사항 확인 가능. 자세한 사항은 mergetool 부분 참고

파일명을 변경한다던지, 삭제한다던지 할 때엔 mv나 rm이 아닌 git mv 나 git rm을 하고 commit을 한 다음 적용해야 remote에도 제대로 적용이 됨

KISTI에 git 깔고 동기화한다음 수정해서 재동기화 예제

git init

git config --global user.name “Yong-Yub KIM”

git config --global user.email kimyy308@snu.ac.kr

git config --global init.defaultBranch main

git config --global alias.unstage 'reset HEAD --'

git remote add ssh https://github.com/kimyy308/SSH\_KHOA.git

git remote -v

git pull ssh main

git status

vi test\_KISTI

git add test\_KISTI

git status

git commit

git push ssh main