GIT == Git Is Team

• Git 설치하기

• Github에 코드 저장하기 - 버전 관리

• Git 변경사항 확인하기

• Git으로 수정사항 reset하기

• Git Branch를 다루는 방법

• Github code versioning

• github에서 협업을 하는 방법

• git playground

Git 설치

Git install (Window)

1 Window Git 설치

2 "**git bash**"를 데스크탑 검색한 후 실행한다.

Git install (mac)

1 현재 git version을 확인한다.

◦ $ git --version

2 Homebrew (Mac OS 패키지 관리자)를 설치한다. 링크

3 Homebrew를 이용하여 git을 설치한다.

◦ $ brew install git

4 설치된 git 정보 확인

◦ $ brew info git 여기서 Dependencies에서 Optiona의 pcre가 (x)로 되어 있다면 pcre도 설치해준다. ($ brew install pcre)

5 설치 후 전과 동일한 버전 정보가 나온 경우 다음을 명령한다.

◦ echo "export PATH=/usr/local/bin:\$PATH" >> ~/.bash\_profile

기본적인 터미널 명령어 정리

$ cd ./myFolder # 현재 폴더 안의 myFolder로 들어간다. ("./"는 생략할 수 있다.)

$ cd ../ # 현재 폴더의 부모 폴더로 이동한다.

$ pwd # 현재 어디 디렉토리에 있는지 경로를 표시한다.

$ ls # 현재 디렉토리 안에 있는 파일 목록들을 보여준다.

$ tree # 현재 디렉토리 안에 있는 tree구조를 보여준다.

# brew install tree로 설치해줘야 한다.

$ mkdir myFolder # myFolder 폴더를 현재 경로에 생성한다.

$ touch myFile.html # myFile.html을 현재 경로에 생성한다.

$ rm -rf myFolder # myFolder를 묻지도 따지지도 않고 다 지우기.

$ rm -i myFolder # 파일 하나하나 물어보고 다 지우기.

$ alias 단축명령어="명령어"

git 설정 (git config)

1 $ git config --global --list

• git에 대한 설정을 볼 수 있다.(전역으로)

• $ git config --list 으로 저장소별로 설정을 조회할 수 있다.

2 $ git config --global user.name github-username

• git user name 설정 (--global의 의미는 해당 디렉터리뿐만 아니라 전체 모든 디렉터리에 적용된다는 의미이다.)

• ex) $ git config --global user.name amamov

3 $ git config --global user.email github-email

• git email 설정

• ex) $ git config --global user.email ysangsuk78@gmail.com

github에 코드 저장하기

git 서버 저장소(원격 저장소) 만들기

1 github에 접속하여 Repositories에 들어가서 New 버튼을 누른다.

2 Repository를 만들어준다.

git 로컬 저장소 만들기

1 git bash (Window OS) 또는 터미널(Mac OS)를 실행하고 작업 디렉터리로 들어간다.

$ cd work-dir

2 git을 초기화한다.

$ git init

3 git local 작업 디렉터리의 모든 변경 사항들을 staging area로 이동시킨다.

$ git add .

4 git local 작업 디렉터리의 특정 파일 또는 폴더의 변경 사항들을 staging area로 이동시킨다.

$ git add filename(or dirname)

5 git staging area에 있는 변경 사항들을 repository로 이동시킨다.

◦ 짧은 commit message로 commit 하는 경우 : $ git commit -m "commit message"

◦ 길고 자세한 commit message로 commit 하는 경우 : $ git commit으로 작성 창으로 넘어간 후 자세히 작성

◦ $ git commit -am "commit message" : add와 commit을 동시에 한다. (한번이라도 commit을 했던 파일에 대해서만 사용할 수 있다.)

$ git commit -m "first commit"

6 main branch를 생성한다.

$ git branch -M main

git 서버 저장소(원격 저장소)와 로컬 저장소 연결 & Push & Pull

1 서버 저장소와 로컬 저장소의 통로를 만든다.

$ git remote add origin https://github.com/amamov/git-study.git

2 로컬 저장소의 repository를 서버의 origin main의 repository에 올린다.

◦ $ git push -fu origin main 명령어는 로컬 저장소의 repository를 서버의 origin main의 repository에 강제로 올린다. (서버 저장소 파일들은 사라진다.)

$ git push -u origin main

4 서버 저장소의 repository를 로컬 저장소에 가져온다.

$ git pull

git 서버 저장소에 있는 Repository를 로컬 저장소에 저장하기

$ git clone https://github.com/amamov/git-study.git

Git 변경사항 확인하기

• $ git log : git에 commit된 변경사항을 확인할 수 있다.

• $ git diff commit\_1\_ID commit\_2\_ID : commit\_1\_ID 기준으로 commit\_2\_ID와 변경사항을 비교한다.

• $ git diff 로컬저장소 원격저장소 : 원격 저장소와 로컬 저장소 간의 변경사항을 비교한다.

◦ ex. $ git diff 비교대상\_branch이름 origin/branch이름

• $ git diff 비교대상\_branch\_이름 기준\_branch\_이름 : 브랜치간의 변경사항을 비교한다.

Git으로 수정사항 reset하기

repository에 올린 상태에서 $ git reset으로 되돌릴 수 있다.

**어디까지 되돌릴까?**

**옵션**

작업 디렉터리가 수정하기 전으로 바뀐다.

--hard HEAD^

add하기 전 상황으로 되돌아간다. (디폴트)

--mixed HEAD^

commit한 것 까지만 취소한다.

--soft HEAD^

• HEAD : 가장 최근 버전에서

• ^ : 하나 되돌리기

만약 가장 최근 커밋으로부터 세 개 전으로 되돌리려면 $ git reset --hard HEAD^^^를 사용하면 된다.

특정 커밋으로 되돌리려면 $ git reset --hard 버전\_ID 명령을 사용하면 된다.

$ git add .으로 stage에 올라간 파일들을 $ git reset HEAD로 취소할 수 있다.

reset 대신에 revert를 사용하면 되돌린 버전 이후의 버전들은 모두 유지되고, revert되었다는 사실을 담은 commit만 새로 추가된다. 즉, reset은 과거 자체를 바꾸는 명령어이고, revert는 과거를 변경시켰다는 새로운 commit으로써 새로운 commit을 만드는 명령이다.

Git Branch를 다루는 방법

• $ git branch

◦ 현재 branch 목록 보기

• $ git branch 브랜치이름

◦ 새로운 branch를 만들다.

• $ git checkout 브랜치이름

◦ 특정 branch로 들어가기

• $ git merge 합치려는\_브랜치\_이름

◦ 현재 브랜치와 합치려는 브랜치를 합친다.

• $ git branch -d 삭제하려는\_브랜치\_이름

◦ 해당하는 브랜치를 삭제한다.

Github code versioning

**git remote (-v)**

원격저장소 **조회(추가)하기**

• 내 로컬 repository와 상호작용하고 있는 (혹은 할 수 있는) 원격 저장소들의 목록을 조회할 수 있다.

• -v옵션 : 단축이름과 URL 같이 보기

• $ git remote add <단축이름> <URL> : 기존 워킹 디렉터리에 새 원격저장소를 추가하는 명령어

◦ $ git remote add origin <URL> : URL에 있는 원격저장소를 origin이라는 이름으로 추가하기

• $ git remote rm <단축이름> : 단축이름의 원격저장소와의 연결을 삭제한다.

**git push**

원격저장소에 **밀어넣기**

• $ git push -u origin main : 내 repository의 main 브랜치를 origin의 main 브랜치로 push한다.

◦ -u : origin main을 디폴트로 지정. 즉, $ git push, $ git pull를 할 때 디폴트로 origin main에 push, pull한다.

**git pull**

원격저장소 **갖고 와서 합치기**

• $ git pull (origin main) : origin을 내 repository의 main 브랜치로 가지고 온다.(merge)

**git fetch**

원격저장소 **일단 갖고만 오기**

• $ git fetch (origin main) : origin을 로컬 repository에 merge하지 말고 일단 가지고 온다.

• $ git checkout origin/main : origin/main 임시 브랜치로 바꾼다. -> origin의 변경된 사항을 확인할 수 있다.

**git clone**

원격저장소 **복사하기**

• $ git clone <URL> : URL에 있는 원격 저장소 내용을 현재 디렉터리에 복사해오기.

◦ origin 자동 생성. 따라서 remote add는 할 필요 없다.

• $ git clone <URL> --branch <branch> --single-branch <folder>

◦ URL에 있는 원격저장소의 특정 branch를 clone하여 folder명을 지정한다.

• git clone시에 origin이 자동으로 생성되므로 이를 없애주려면, $ rm -rf .git 명령어를 사용하면 된다.

github에서 협업을 하는 방법

joy라는 팀장과 amamov라는 팀원이 협업을 한다고 가정하자. 팀장이 만든 repository인 프로젝트 REPO를 중심으로 협업을 진행한다고 하자.

amamov는 팀장이 만든 repository를 clone한 후, 로컬의 "amamov/main" branch에서 팀장의 코드를 받아오고 "amamov/newbranch" branch에서 받아온 코드(main)를 병합하여 추가로 작업한 코드와 함께 팀장에게 pull request를 보낸다. pull request를 받은 팀장은 변경 사항을 체크하고 "joy/main"에 최종적으로 병합한다.

**1. 협업 대상 repository fork 하기**

amamov는 joy/REPO에 들어가서 fork 버튼을 누른다.

**2. fork 해온 곳에서 clone하기**

amamov는 자신이 fork한 repository를 clone한다.

**3. branch를 만들고 작성하고자 하는 코드(commit) 편집 후에 push**

1 amamov는 clone한 REPO에 들어가서 $ git branch newbranch 명령을 통해 "newbranch"라는 branch를 만든다.

◦ "amamov/main" branch는 팀장의 코드를 가져오는 순수 branch이다. (amamov가 추가적으로 코드 작업을 하지 않고 순수하게 팀장의 코드를 가져온 branch)

◦ "amamov/newbranch" branch는 팀장의 코드와 amamov가 추가적으로 코드 작업을 진행한 branch이다.

2 $ git checkout newbranch 명령으로 "newbranch"에 들어간다.

3 amamov는 newbranch에서 코드를 작성하고 파일을 만드는 등 작업을 한다.

◦ $ git add .

◦ $ git commit

▪ i를 누르고 내용 작성 후에 esc키를 누르고 :wq!명령어로 저장하고 나간다.

◦ $ git push origin newbranch

▪ "newbranch"를 origin에 push한다.

**4. pull request**

1 amamov는 자신이 fork한 REPO repository에 들어가기 위해 github에 들어간다.

2 github의 해당 repository에 들어가서 Compare & Pull request 버튼을 누른다.

3 그 다음 Create pull request를 눌러서 pull request를 시도한다.

**5. Merge pull request**

1 탐장은 github에 amamov가 pull request한 amamov:newbranch가 보인다.

2 만약 충돌이 발생할 경우 충돌을 해결한다.

3 Merge pull request 버튼으로 병합을 받아준다.

**◦ revert**를 이용하여 병합 이전으로 돌아갈 수 있다.

4 (관리를 위해 amamov는 newbranch를 지우는 경우가 있다.)

**6. 팀장(repository 관리자)이 REPO repository를 수정하였을 때 amamov(팀원)가 팀장이 수정한 "joy/main"을 가져와야 한다.**

amamov는 자신의 로컬 환경의 REPO에서 다음과 같은 작업을 수행한다.

1 $ git remote add upstream <joy-URL> **(최초 한 번만)**

◦ "upstream"이라는 이름으로 joy의 repository URL을 remote한다.

2 $ git fetch upstream

◦ upstream에 fetch한다.(일단 가져온다.)

◦ 변겨 사항 확인 가능

3 $ git checkout main

◦ "main" branch로 들어간다.

4 $ git merge upstream/main

◦ "upstream"과 main을 병합한다.

5 $ git checkout newbranch

◦ 작업용 branch인 "newbranch"에 들어간다.

6 $ git merge main

◦ "main" branch와 현재 branch인 "new branch"를 병합한다.