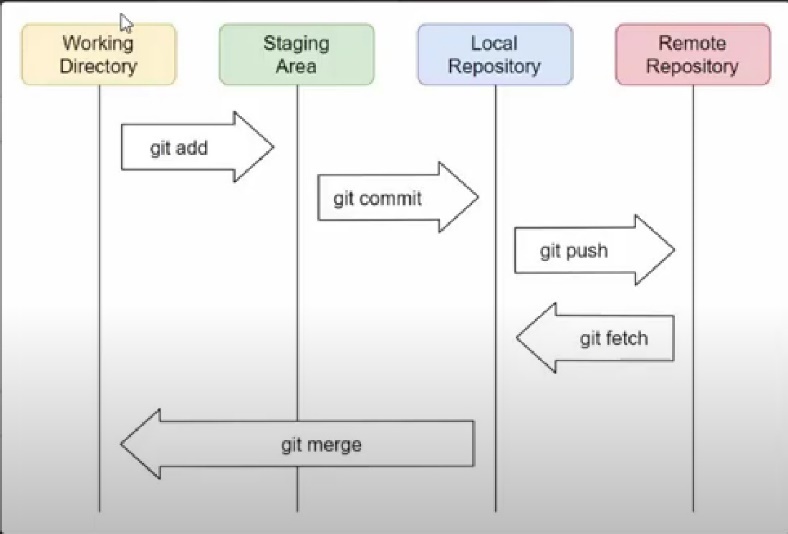
git 공부

깃의 동작 원리

Working Directory 작업할 파일이 있는 디렉토리

staging Area 커밋을 수행할 파일들이 올라가는 영역(git add명령을 하면 이곳으로 간다)

git directory 깃 프로젝트의 메타 데이터와 데이터 정보가 저장되는 디렉토리



깃 커밋까지는 내 로컬상에서의 작업이다 [.git]폴더에 작업사항들이 기록된다.

깃 푸쉬를 통해서 원격 저장소에 내용을 저장하게 된다.

원격저장소에서 변경된 내역은 (깃 패치)를 통하여 로컬 레포지토리로 받아올 수 있다

깃 패치 - 깃 머지를 통하여 원격 저장소에 있는 내용을 로컬로 받아올 수 있다.

깃 패치 + 깃 머지 = 깃 풀

- 소스코드를 수정하는 사례 두 가지

1.어떠한 Community의 구성원이 아니어서 스스로 커밋을 하여 저장소에 적용할 권한이 없다면

소스코드를 수정하는 것에 제약이 있다. 이런 경우 PR(Pull Request)를 작성하여 오픈소스에 기여 가능

2. 프로젝트에 대한 권한을 가지고 있다면 Commit하고 Push해서 저장소에 수정 내역을 반영하면 됨

git clone의 경우 남들이 만들어 놓은 것들도 다운로드할 권한이 있다면 내 로컬로 받아서 이용 가능하다.

주의: 깃을 cmd로 사용하려면 깃 사이트에 가서 깃부터 받아야합니다~

git: 깃 기본 명령어 정리

git --version 깃 버전 정보

git config --global

git config --global user.name 깃허브아이디

git config --global user.email 깃허브등록이메일

cd c:\복사하려는 위치 (연습용위치 cd c:\Education)

git clone 주소(깃허브에서 복사한 https주소)

온라인 상의 레포지토리를 내 로컬의 에듀케이션 폴더로 복사한다.

해당 디렉토리로 와서

git add 넣을 파일 git add document 도큐먼트라는 파일을 staging area에 올리겠다.

(내리려면 gti reset 파일명 하면 된다. 에드해서 스테이징 구역에 올려야 커밋 가능

중요 기능!! git add . 을 하면 특정 파일명을 입력하지 않더라도 수정된 내용을 에드 가능하다.)

git commit -m "Add Text File [document]" <업로드 된 그 시점을 스냅샷처럼 남긴다.

나중에 문제 생길 시 복구하는 시점으로 돌릴 수 있다. (커밋 됨)

git push [커밋된 내용을 원격 저장소로 밀어 넣기]

git status 현재 디렉토리에서 변경된 사항들을 표시한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Tutorials 폴더 안에서 my\_module.py라는 파일을 만들었다.

git status를 입력하니 빨간색으로 저것이 추가되었다고 표시된다. (커밋 될 수 있다고 표시 됨)

git commit -m “메모내용” 통해서 커밋을 할 수 있다.

git push를 통해서 원격 레포지토리에도 업데이트를 해준다.

자료 추가해서 올리기

MyEx라는 파일안에 글자를 추가한 뒤에 연습하기

다시 git status를 실행하면 변경된 내역이 표시된다.

git checkout - - 파일명(마이너스 두개 이어 붙여쓰기) 하면 변경된 내역이 무시된다.

해당 파일을 열어보면 변경되었던 내역이 사라져있다.

커밋할 때 “” 안의 설명을 변경하고 싶을 때는

git commit - -amend입력해서 수정한다. (뒤에서 자세히 설명)

변경된 내역은 git log 명령어를 통해서 로그를 확인 가능하다. q 누르면 나가짐

git log는 위에 출력된 값이 최신 값이다.

커밋은 모두 해쉬값으로 지정되어 있다. 해당 해쉬값을 이용하여 특정 시점으로 돌아갈 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Head->main 이라고 적힌 것은 현재 checkout되어 있는 브랜치의 가장 최근 commit을 가리킴(현재 해당 브런치에

체크인 되어있고 최근 커밋도 함)

노란색으로 보이는 것이 특정 커밋 에 대한 해쉬값이다.

git reset - -hard 해쉬값

실행 시에 해당 커밋 이후에 커밋된 건은 싹 날려버린다. 하드한 옵션이라고 할 수 있다.

git reset - -hard 3425e8898614d7c8c59fa42043a6a3ae1a59c78d 실행 후(마지막 커밋 직전 시점으로 돌아가기)

로컬은 마지막 커밋을 실행하기 전의 상태로 변경되어 있는 것을 확인할 수 있다.

다만 이 상황은 내 로컬만 변경이 된 상황이다. 원격로컬에도 변경사항을 적용하기 위하여

git push를 실행해보면 rejected 당하는 것을 알 수 있다.

이때는 git push -f 명령어를 통하여 강제로 원격 레포를 내 로컬과 동일하게 만들 수 있다.

(협업시에는 사용하면 큰일 날 수 있다.)

커밋 메시지 변경(위에서 자세히 설명한다던 그것)

텍스트이(가) 표시된 사진

자동 생성된 설명

git commit -m “Add MyEx.txt [메세지]”에서 메시지 내용을 바꾸기 위해서

git commit –amend 명령어를 사용할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

유닉스 에디터라고 하는 것이 실행된다. [수정모드/관리자모드 있음]

수정모드로 들어가기 위해서 a를 입력한다. [수정모드 일 때만 글씨를 수정가능]

내용을 바꾼 뒤에 esc 누르고 wq! 를 입력한다. 그러면 수정된 상태로 밖으로 나온다.

텍스트이(가) 표시된 사진

자동 생성된 설명

[add]에서 [Add & Minus]로 변경되었음을 확인할 수 있다.

깃 브랜치 (git branch)

깃 저장소를 만들면 자동으로 마스터 브랜치가 생성된다.

마스터 브랜치는 항상 안정화되어 있어야 하고 언제나 배포 가능한 상태여야한다.

개발시에는 버그 픽스, 브랜치 개발 브랜치 등을 따로 만들어둔다.

통합 브랜치: 배포가 가능한 수준의 브랜치로 일반적으로 마스터브랜치라고 한다.

토픽 브랜치: 특정 기능을 위해 만들어진 브랜치로 일반적으로 마스터 브랜치 이외의 것.

git branch 명령어 -> 현재 있는 브랜치들을 보여준다.

git branch 브랜치명 -> 해당 이름의 브랜치를 만든다.

git checkout 브랜치명 -> 해당 브랜치로 브랜치를 변경한다.

git merge 통합할 브랜치명

실습

브랜치를 사용할 때는 시작시에 항상 pull을 받아서 최신 상태임을 확인하고 시작한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

로그 안에 main과 브랜치 exp 모두 표시되는 상태임을 확인 할 수 있다.

Djdevelop 브랜치를 만들어서 작업하기

1. 브랜치를 만든다. git branch Djdevelop
2. 파일을 변경한다.
3. Djdevelop에서 커밋까지 마무리한다.
4. 텍스트이(가) 표시된 사진

   자동 생성된 설명git log로 확인해보면 최근에 Djdevelop 브랜치를 이용한 커밋에 HEAD가 Djdevelop를 향하고 있음을 알 수 있다. 브랜치 별 로그 확인 -> git --branches --decorate
5. git checkout main으로 메인 브랜치로 돌아간다.
6. git merge Djdevelop 명령어로 두 브랜치를 통합한다. [통합은 메인에서 하자!!!]
7. git push통하여 원격 Depo에 변경사항을 반영한다. 텍스트이(가) 표시된 사진

   자동 생성된 설명반영 이후에 git log를 실행해보면 위 사진과 다르게 HEAD가 모두 42b212f를 가리키는 것을 확인할 수 있다.

깃 브랜치 제거하기 -> git branch -d 브랜치명

충돌 처리하기 (conflict)

동일한 파일을 병합할 시에 충돌이 일어날 수 있다. 충돌을 해결하는 방법을 알아보자

상황 git develop에서 MyEx.txt 내용 변경 후 add -> commit 진행함

깃 브랜치를 메인으로 변경한 뒤에 MyEx.txt 파일을 다르게 변경한 후 커밋까지 완료한다.

이제 git log를 확인해보면 차이점이 보인다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다시 깃 develop 브랜치로 변경한 뒤 git log를 실행한 화면이다.

깃 main에 대한 사항이 표시되지 않는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다시 메인으로 돌아와서 로그를 확인했다. 역시나 main밖에 표시되지 않는다.

이전에는 같은 파일이기 때문에 메인과 다른 브랜치가 로그에 표시되었지만

이제는 둘이 완전 다른 파일이 되었음을 의미한다.

이 때 머지를 시도하면 다음과 같이 뜬다.

텍스트이(가) 표시된 사진

자동 생성된 설명

자동 병합 실패: 충돌을 수정하고 결과를 커밋하라.

이 때 원본파일을 열어보면 이렇게 표시가 되어있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

=== 위는 메인 아래는 develop의 코드이다.

메인쪽을 선택해서 충돌을 수정해본다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이것만 남기고 ===포함하여 삭제한다. 저장하고 파일을 닫는다.

add 후에 커밋한다.

git merge develop 명령어를 사용해보면 Already up to date 가 뜬다 이미 수정된 것

git log를 살펴보면 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

한 log안에 메인과 develop가 모두 있는 것을 확인할 수 있다. 같은 파일이 된 것.

git push를 통해서 원격지 저장소에도 변경내용을 반영한다.

브랜치 작업에 대한 심화 내용은 아래 링크 참조

https://codingmomong.tistory.com/22

일상 협업 루틴

원본파일과 너무 동떨어지지 않도록 주기적으로 pull을 해줘야 한다.(main과 내 브런치 동기화)

마스터에 변화가 감지가 된다면 모든 작업자는 pull해주는 것이 좋다.

동기화 할 땐

일단 git add . -> git commit -m “커밋내용” 해서 작성하던 것까지 커밋한다.

여기에서 푸쉬 하지 말고 git pull origin main을 통하여 메인으로부터 최신본을 받아온다

이후 작업이 완료되면 push해준다.

각종 문제 사항 발생 시 해결 방법

1번) git add . 을 했는데 warning: LF will be replaced by CRLF in 으로 시작하는

경고문이 뜨는 경우 : git config --global core.autocrlf true 입력 (참고 <https://dabo-dev.tistory.com/13>)

2번) 다음과 같이 무언가 파일이 스테깅 지역에 올라가지 않는 경우

텍스트이(가) 표시된 사진

자동 생성된 설명

git add Project2/.metadata/.plugins/org.eclipse.e4.workbench/workbench.xmi 와 같이 문제되는 부분을 직접 add에 집어넣어서 올리면 잘 되는 것 같다.

3번) 만약 올라가야 할 부분이 인식이 되지 않는 경우에는?? 정작 올라가야 할 프로젝트는 내용들은 감지가 안되고 메타데이터들만 잔뜩 업데이트 되는 경우에는 직접 강제로 add시켜준다 -> git add --force C:\PF0522\portfolio\Project2/ 해당 경로의 preject2이하의 것들을 모두 강제로 add하겠다. 무시되는 것들을 강제로 add 할 것이라는 의미

참조 사이트 및 강의

나동빈 git 유튜브가이드 터미널을 이용한 깃 사용 기초부터 협업까지

<https://www.youtube.com/watch?v=rhP5pseOJc0&list=PLRx0vPvlEmdD5FLIdwTM4mKBgyjv4no81&ab_channel=%EB%8F%99%EB%B9%88%EB%82%98>

브랜치 작업에 대한 심화 내용 블로그

https://codingmomong.tistory.com/22

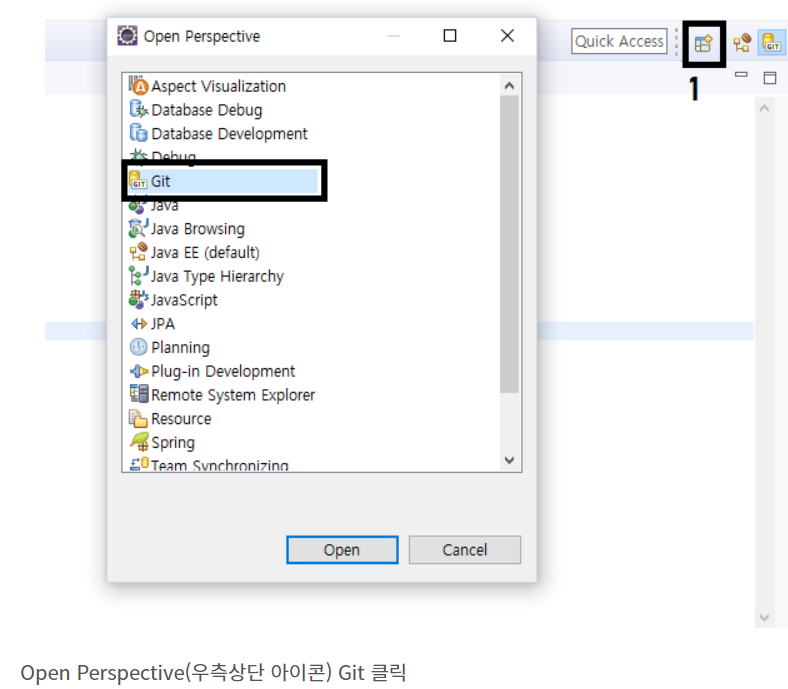
회사에서 깃 사용하는 대략적인 프로세스를 설명하는 유튜브 (코딩알려주는누나)

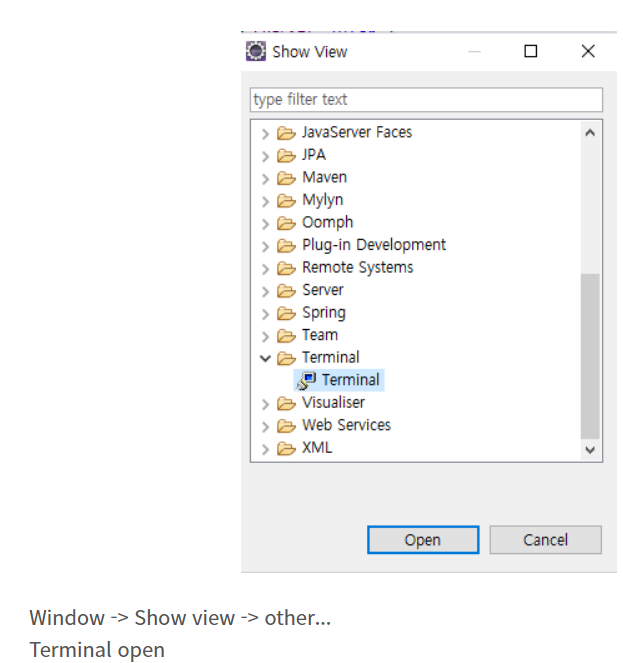
https://www.youtube.com/watch?v=cwC8t9dno2s&ab\_channel=%EC%BD%94%EB%94%A9%EC%95%8C%EB%A0%A4%EC%A3%BC%EB%8A%94%EB%88%84%EB%82%98

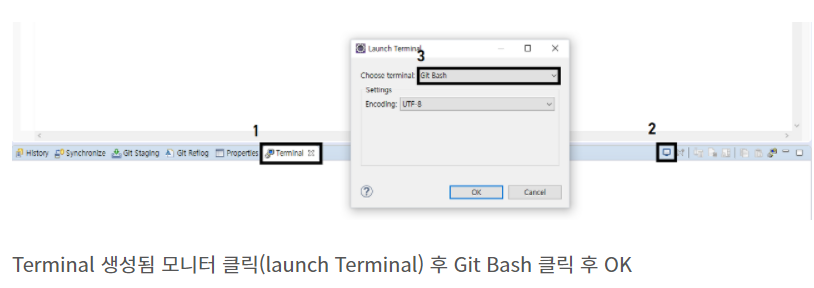
추가 유용 블로그 정보 https://parksb.github.io/article/28.html

번외편

Eclipse에서 Git 사용하기







텍스트이(가) 표시된 사진

자동 생성된 설명show view 등에서 package explorer 불러와서 세팅하고 오른쪽 상단에 open perspective에서

Java ee도 추가해준다. 이렇게 세팅을 해주면 이미지처럼 cmd를 따로 사용하지 않아도 이클립스내에서 작업이 가능하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

깃 브랜치 작업 심화 내용

git log –-branches --decorate –graph 마스터와 다른 브랜치가 각자의 길을 걷고 있을 때 활용

텍스트이(가) 표시된 사진

자동 생성된 설명git log --branches --decorate

--graph --oneline

윗 내용을 한 줄로 간편하게

알려준다.

텍스트이(가) 표시된 사진

자동 생성된 설명git log main ..DJ2

git log 브랜치1 ..브랜치2

브랜치1과 2의 로그를 비교하기

git log -p main ..DJ2 하면 소스코드

차이까지 알려준다.

git diff main DJ2 두 브랜치 간의 현재 상태를 비교

깃 브랜치 병합

브랜치에서 작업을 한 이후에 main으로 체크아웃한다.

깃 충돌시험 12