git – 생활코딩

|  |
| --- |
| git1  깃의 3대 목적.  버전관리, 백업, 협업  (순서) 뒤가 앞에 의존. 어려워.  폴더에 깃 저장소 연결.  커밋하면 로그가 기록된다.  변경한 상태에서 이전과 비교 가능.  커밋하면 플래그 생성.  변경하고 커밋 안하고 도 변경하면 플래그는 없겠지.  내 컴퓨터에만 보관하면 언젠가는 그 파일이 유실될 수 있다.  그게 아니더라도 나는 노트북이 2대니까 이곳저곳 저장이 분산되게 된다.  원격 저장소. 로컬 저장소.  깃허브 사이트 – 내 컴퓨터.  깃허브에 저장한다. push  깃허브에서 가져온다. pull.  [커밋와 푸시의 차이는?]  커밋은 버전이 생성된다.  커밋하면 버전이 변경되었다며 push가 뜬다.  push를 누르면 버전이 저장되는 구조인가보다.  원격 저장소를 통해 push와 pull을 하며 협업이 이루어진다.  깃 허브 데스크탑. 프로그램.  git bash. 사용해보기.  git log [버전 로그 보여준다.]  git status [변경사항 알려준다.]  git commit -am "message" [커밋.]  [-m은 아는데 a는 뭐지?]  git push [업로드] |
| 파일 A,B를 수정했다.  파일 A만 따로 버전 관리 가능.  버전 관리하고 싶은 파일을 선택 – 스테이지에 올린다.  스테이지에만 올린 상태 – stage area  워킹 카피/디렉터리. 물리적 개념 – 폴더와 파일들.  stage area는 논리적인 개념인가보다.  정보 시스템에 따져봐야 할 것.  CRUD.    reset 버전 돌리기.  클릭한 상태로 되돌아가겠다는 뜻.  mixed나 soft를 선택하면 변경 안된다고 한다.  hard를 선택하자.  위험성.  인터넷-원격에 올라가 배포한 버전은  reset하면 안된다고 한다.  reset은 삭제하는 것.  삭제하지 못하는 경우가 있다.  기록을 실수해서 지우고자 한다.  그런데 지우면 안된다. 실수조차 기록..  어떻게 이 문제를 해결할 수 있을까?  실수했다고 기록하는 모양이다. 과정은 복잡하겠지만  이것이 revert.  revert는 reset보다 어렵다.  되돌리기.  함부로 reset하는 것 좋지 않다.  실수조차도 기록하기 revert.  오른쪽 마우스.  reset은 되돌아가고 싶은 버전을 클릭한다.  revert == reverse는 실수=기록을 지우고 싶은 버전을 클릭한다.  자동으로 커밋이 생성된다.    revert7 되어 있는 상태에서 M4로 revert하고 싶다?  바로 M4 revert 하면 안된다.  충돌이 일어난단다.  그럼 어떻게 할까?  revert 6과 5를 하면 M4가 된다.  이렇게 해야 한단다.  revert는 역순으로 하나씩 해나가는 거라는데..  최초에 할 때,도 겠지?  M1-7이 있는 상태에서?  M4를 revert하겠다는 의미는,  M4 이후의 모든 변경사항을 버리겠다는 뜻이 아니라고 한다.  M4에서 일어난 변경사항만!을! 되돌리겠다는 의미라고 한다.  그런데 M4만 지우면 그 이후의 사항에 영향을 주기 때문에  깃이 conflict 충돌을 발생시킨다고 한다.  단순히 과거의 상태를 보고싶은 거라면  더블클릭하면 된다고 한다.  그러면서도 버전을 삭제되지 않고 남아있다.  가장 최신 상태로 돌아가고 싶으면  왼쪽 네이게이션의 master를 더블클릭하면 된다.  왜 그런지는,  나중에 브런치, 헤드를 배우면 알게된다.  각각의 버전은 커밋이라고도 부른다.  각 커밋은 고유한 식별자가 있다. commit id, sha1  버전id, 리비전id  버전관리의 장점은.  변경 이전과 이후의 차이를 쉽게 비교할 수 있다는 점.  텍스트 파일은 아주 쉽게 확인할 수 있다.  그런데 한컴, 이미지 등은 비교가 어렵다.  비교할 수 있는 방법을 살펴보자.  스테이지에 올리기 전.  오른쪽 마우스. Open Current Version.  밑에 Open Selected Version 클릭.  각 버전에서 창을 열고 비교 가능하다.  외부 도구를 이용해서 비교하는 방법도 있다.  KDiff3 프로그램.    두 파일의 차이를 보여준다. 편리.  파일 이름 봐봐.  이런 프로그램을 소스트리와 연결할 수 있다.  소스트리 위쪽 네비게이션 [옵션]    프로그램 설치 후 선택하면 된다.  항목에 없는 것도 가능하다고 한다.    이전 상태와 비교하고 싶은 파일을 선택하고 (위) 클릭하면  연동된 프로그램으로 비교 가능하다고 한다.  음.. 한 번 해보면 알거같다.  버전 관리의 핵심은 비교.  비교를 통해서 과거를 되돌아볼 수 있다는 것이  버전 관리의 핵심적인 효용.  diff tool 검색하면  여러 비교 도구들이 있다고 한다.  차이점을 정교하게 비교하고  현재 상황을 신속하게 파악하는 데 정말 큰 도움이 된다고 한다.  버전 관리하지 않을 파일이 있을 수 있다.  .gitignore 파일을 만들고 거기에 파일 이름을 적으면 된다.  또 깃의 정말 중요한 특징,  깃의 혁신적인 면 중 하나는 branch.  마치 평행우주처럼  우리 저장소를 여러 가지 상태로 공존할 수 있게 해줍니다.  브랜치를 이용하면  저장소의 이름을 더럽히지 않고  하나의 저장소에서 다양한 작업을 진행할 수 있다.  사람이 보기 힘든 commit id에 대응하는 것이 tag  백업도 중요하다.  깃은 자체적으로 매우 편리하고 안전한 백업 장치를 가지고 있다.  나는 노트북이 두 대니까  깃으로 백업 연습을 해보면 좋을것 같다.  비공개 계정으로 해보자!  Sisi55는 공개 계정.  siCsds3은 비공개 계정.  백업에 익숙해지면  자연스레 협업이라는 기능이 보인다. |
| 깃 호스팅이란 작업하고 있는 로컬 저장소에 버전을 업로드할 원격 저장소를 임대해주는 비즈니스.  잘 이용하면 백업 서버인 원격 저장소를 운영할 수 있다.  복제. 클론.  원격 저장소와 지역 저장소가 같은 상태가 된다.  GitLab. 오픈소스 프로젝트에 대해서는 무료.  오픈소스라서 우리의 서버에 직접 설치할 수 있느냐.  GitLab은 private 저장소를 무제한으로 생성할 수 있다.  그래서 GitHub와 GitLab을 같이 사용하자.  GitLab에서도 저장소를 만들었다.  앞으로 어떻게 공부할 것언지 전략을 세워보자.  어떻게 지역 저장소와 원격 저장소를 연결할 수 있지, 하는 궁금증.  통신 방법 2가지.  http, ssh  우리는 http를 배운다.  http는 보안적으로 조금 부족하다. 조금 불편할 수도 있다.  근데 배울 필요가 없다. 그냥 할 수 있다.  ssh는 보안쪽으로 편리하고 강력하다.  하지만 배워야 할 것이 꽤 많다.  그리고 혼란스럽다.  원격 저장소와 지역 저장소를 연결하는 방법을 살펴봅시다.    [리파지토리]-[add remote] 선택하고 저장소 url 복붙.  기본적인 원격 저장소는 origin으로 하기로 약속되어 있다.  (그래서 origin이었구만)  지역 저장소에서 작업한 내용을 여러 저장소에 분산해서 필요에 따라 업로드 할 수 있다.  별명을 통해 저장소를 구분한다. ? (잘 모르겠군)  (내가 add remote를 해놓았던 것 같기도 하다.. 이거저거 하다가)  원격 저장소로 업로드 하는 push. |
| push를 누르면 나오는 창.  local branch가 master, remote branch가 master.  연결시켜서 올리겠다. 이게 track.  branch를 알면.. 일단 체크하고 넘어가자.    최초에 누구인지 물어보는 창이 나왔다고 한다.  내가 잘못 입력했던 듯.  github 등의 계정 id, password를 입력하면 된다. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |