git 기본정보

git 소개

- 분산파일 시스템

subversion : 파일변화를 시간단위로 관리

git : 스냅샷형태로 관리

commited : 데이터에 DB에 저장

modified : 수정한 파일이 DB에 커밋되지 않음

staged : 수정한 파일을 커밋

working tree의 파일 수정

staging : 커밋한 스냅샷 생성, staging에 커밋해서 git에 업로드

tracked : git이 관리하는 파일

  - unmodifed : 수정되지 않은 상태

- modified : 수정된 파일

- staged : 커밋후 저장소 기록예정

untracked : git이 관리하지 않는 파일

author : 원작자

committer : 수정자

git add를 복구할때

  -> git reset HEAD 파일명

리모트 저장소는 origin이 자동등록

git remote add {name} URL

git fetch {name}

 --> origin으로 입력하면 clone이후 수정한 모든 데이터를 가지고 옴

리모트 저장소에 push방법

-> git push origin master

-> 다름사람이 push한내용은 push 불가

-> merge 후 push 가능

태그 : 2가지 형태의 태그 기능

light weight : 특정커밋의 포인터 이름

-> git tag {tagname}

annotated : 일반적인 태그 방법

-> git tag -a {tagname}

특정커밋의 태그 확인

-> git log --pretty=online

태깅할 체크썸 확인

-> git tag -a {tagname} 체크섬 전부 다 입력할 필요는 없고 앞 7자리만 알고 있으면 됨(고유번호)

태그 공유

-> git push origin {tagname}

* 여러개인 경우 git push origin --tags

특정 태그 checkout

-> git checkout {tagname}

* detached HEAD(떨어져나온 HEAD) 메시지 출력(정상)

* 브랜치에서 작업하는 것과 다르게 작동하게 됨

커밋하게 되면 생성된 커밋을 사용할 수 있는 브랜치가 다르기 때무네 가급적 브랜치를 이용해서 관리

브랜치

staging area : 스냅샷에 대한 메타 데이터, 이전커밋의 포인터 정보 저장

파일을 stage하면 저장소에 파일 저장(이를 Blob라고 호칭)

기본적으로 master 브랜치 생성된

브랜치 생성 -> git branch {branchname}

브랜치 이동 -> git checkout {branchname}

프랜치 생성하면서 체크아웃 -> git checkout -b {branchname}

머지

git merge {branchname}

* fast-forward : 브랜치 포인터는 자동으로 최신커밋이동

브랜치 삭제 : git branch -d {branchname}

만약 개발자가 개발중에 이슈처리를 하게 되는 경우

git checkout {branchname}

이슈 처리 후 머지하는 경우

git checkout master

git merge {branchname}

머지 충돌

3way merge으로 수행하는 경우 두 브랜치에서 같은 파일을 수정할때 발생

<<< --> 이건 HEAD버전

>>> --> 이건 현재 버전

브랜치의 머지 여부 확인 git branch --merge / --no-merged

머지하지 않은 브랜치 삭제는 -D 사용

브랜치 관리 전략

3way merge이기 때문에 자이간에 걸쳐서 한브랜치를 다른 브랜치와 여러번 머지하는것이 쉬움.

Revision #1
★Created 7 June 2022 02:03:53 by artop0420
✎Updated 24 December 2023 02:21:40 by artop0420