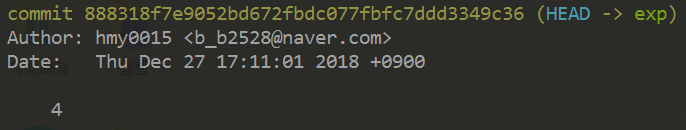
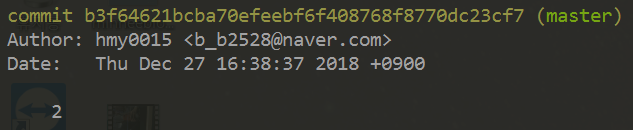
git의 혁신 – branch

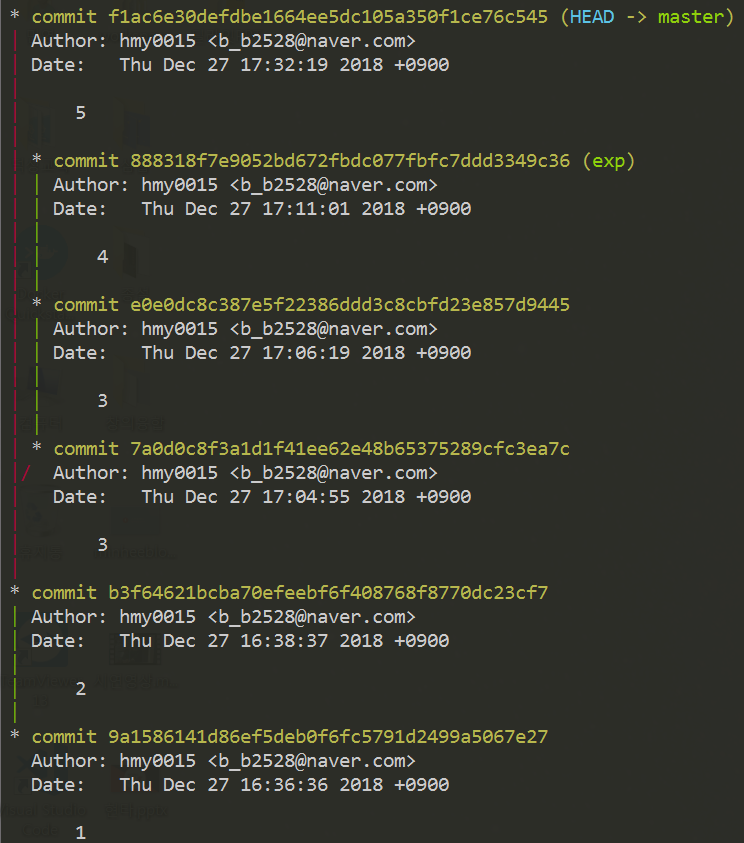
1. branch 만들기
   * git commit -a 사용 시 주의점
     + 1. 한 번도 add 명령을 실행하지 않은 파일은 이 명령을 사용해도 자동으로 add 명령이 실행되지 않음
   * **branch**를 사용하면 **본래 소스코드를 변경하지 않으면서 기능들을 추가**할 수 있음
   * git branch
     1. branch 목록을 보여주는 명령어
     2. \* : 현재 사용되고 있는 branch의 위치를 나타냄
     3. master branch : git에서 사용되는 기본적인 branch
   * **git branch + branch name**
     + 1. **branch를 생성**하는 명령어
       2. 생성된 branch는 사용자가 속해 있는 branch의 소스코드를 그대로 복제하여 생성됨
   * **git checkout + branch name**
     + 1. 사용하고자 하는 **branch의 위치로 이동**
       2. **git checkout -b + branch name** => branch를 **생성**하고, 생성한 branch로 **이동**까지
   * **git branch -d**
     + 1. **branch를 삭제**하는 명령어
       2. **git branch -D** => **강제삭제**
2. branch 정보확인
   * **git log --branches --decorate**
     1. 저장소의 **모든 branch의 기록을 보여줌**



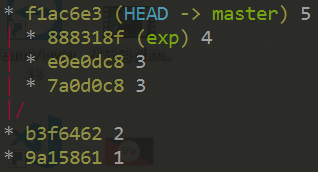
* + - 1. exp branch의 최신 commit은 “4”
      2. HEAD -> exp : 현재 속한 branch의 위치를 나타냄



* + - 1. master branch의 최신 commit은 “2”
    1. **git log --branches --decorate --graph**

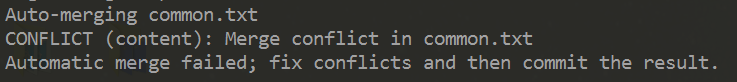


* + - 1. **여러 branch의 commit 상태가 다를 때** 용이
    1. **git log --branches --decorate --graph --oneline**

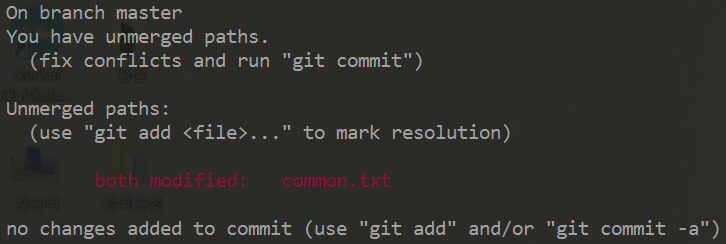


* + - 1. 현재의 **상태를 한 줄로** 볼 수 있음
    1. **git log “비교할 branch 1”..”비교할 branch 2”**
       1. **branch 간에 비교할 때**
       2. branch1에는 없고 branch2에는 있는 것들을 보여줌
* 반대로 쓰면 branch2에는 없고 branch1에는 있는 것을 보여줌
  + - 1. **git log** **–p** “비교할 branch 1”..”비교할 branch 2” : **각각의 버전 별로 어떤 차이가 있는지**를 보여줌
    1. **git diff “비교할 branch 1”..”비교할 branch 2”**
       1. branch 간의 **코드를 비교할 때**

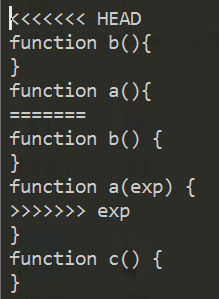
1. branch 병합
   * ‘A’ branch에서 ‘B’ branch를 병합할 때 (A <- B)
     1. **git checkout A**
        1. ‘A’ branch로 이동한 후,
     2. **git merge B**
        1. 병합하고자 하는 branch를 merge 해 줌
2. branch 병합시 충돌해결
   * 충돌발생



* + - 1. 충돌 발생시 위와 같은 메시지가 출력 됨
  + 충돌탐색



* + - 1. git status 명령 실행하면 충돌이 일어난 파일을 찾을 수 있음
  + 충돌해결



* + 1. ‘<<<<<<< HEAD’부터 ‘=======’ 사이의 구간 : 현재 checkout 된 파일의 내용
    2. ‘=======’부터 ‘>>>>>>> exp’ 사이의 구간 : 병합하려는 대상의 branch 코드
    3. 두 정보를 참고로 해서 두 개의 코드를 병합한 후 특수 기호들을 제거 => 파일 저장
    4. 충돌 작업이 끝났음을 git에게 알려줌
       1. git add ‘conflicted file name’
       2. ex : git add common.txt

1. stash
   * stash
     + 1. 아직 현재 branch에서 작업이 끝나지 않은 경우 stash를 이용하면 작업중이던 파일을 임시로 저장해두고 현재 branch의 상태를 마지막 commit의 상태로 초기화 할 수 있음
       2. 다른 branch로 이동해 작업을 끝낸 다음, 작업 중이던 branch로 복귀하여 이전 작업 내용을 복원할 수 있음
       3. 추적되고 있는 작업만 수행됨
   * **git stash --help**
     + 1. stash에서 사용되는 옵션을 볼 수 있음
   * **git stash**
     + 1. stash 실행
   * **git stash list**
     + 1. 숨겨둔 작업 내용을 볼 수 있음
   * **git stash apply**
     + 1. 숨겨두었던 작업 내용을 복원
       2. stash list의 제일 위의 내용을 복원함
   * **git stash drop**
     + 1. 가장 최신 stash 삭제
   * **git stash pop (=git stash apply; git stash drop;)**
     + 1. 작업 내용 복원 후 삭제
   * **git reset --hard HEAD**
     + 1. HEAD => 가장 최근 commit 상태

원격 저장소 – My Server

1. 자기 서버에 원격 저장소 만들기(My Server)
   * 특정 서비스에 의존하지 않고도 저장소를 운영할 수 있음
   * git remote add origin ssh://아이디@서버로 사용할 컴퓨터의 아이피 주소/저장소의 경로/
   * git push --set-upstream origin master
     + 1. 지역 저장소의 master branch가 원격 저장소의 master branch를 연결함
   * git clone ssh://아이디@서버로 사용할 컴퓨터의 아이피 주소/저장하고자 하는 저장소의 경로/
2. push & pull (My Server)
   * **git pull**
     + 1. 저장된 작업 내용을 내려 받음
   * **git push**
     + 1. 작업 내용을 저장소에 저장함
       2. 작업한 내용을 정상적으로 push 하기 위해서는 **작업을 시작하기 이전에 pull을 통해 기존 작업 내용을 내려 받아야 함**
   * push와 pull은 자주 해줘야 **충돌이 일어날 위험을 낮출 수 있음**
3. 자동로그인 (My Server)
   * cd ~ : home directory로 이동
   * /.ssh 파일은 다른 곳으로 이동해 백업
   * ssh-keygen -t rsa
     + 1. rsa는 ssh를 통해 서버에 접속할 때 사용할 암호화 방식 중 하나
   * **id\_rsa - private key**
     + 1. keygen을 통해 만들어 낸 아이디는 복잡한 암호로 이루어짐
       2. **private key는 누구에게도 노출되어서는 안 되는 key**
       3. 소유자에 대해서만 읽기와 쓰기가 가능 (.ssh도 마찬가지)
   * **id\_rsa.pub - public key**
     + 1. 로그인하고자 하는 컴퓨터에 public key를 심어 놓으면 **private key를 가진 사용자가 id/pw를 타이핑하지 않아도 자동로그인** 할 수 있음
       2. 소유자, 그룹, 이를 제외한 타인도 읽을 수 있음

* private key를 통해서 열리는 것을 허용하고자 하는 서버 컴퓨터에 public key를 심어 놓으면 private key가 있는 사용자는 public key를 열 수 있음

tag

1. tag 기본 사용법
   * tag는 고정된 특정한 commit 버전을 가리킴
     + 1. branch는 가리키는 commit 버전이 계속해서 변동됨
   * git tag
     + 1. 태그 목록 보기
   * **git tag + tag name + 태그가 가리킬 버전의 commit 아이디**
     + 1. **light weight tag 생성**
   * git checkout + tag name
     + 1. 해당 태그가 가리키는 commit으로 이동할 수 있음
   * **git tag -a + tag name + -m “태그에 대한 설명” + 태그가 가리킬 버전의 commit 아이디**
     + 1. **annotated tag 생성**
       2. git tag -v tag name 명령 실행 => 태그에 대한 주석을 확인할 수 있음
   * git push --tags
     + 1. 태그를 저장소로 업로드
   * git tag -d “삭제할 태그명”
     + 1. 태그 삭제