Git Study 기초 개념 및 실습자료

v2021-11-03

-by Meteor

전체 목차

Git이란 ? ? ?

Git이란 분산형 버전 관리 시스템(Version Control System)의 한 종류이며, 빠른 수행 속도에 중점을 둔다.

예를들어 ppt를 만들 때 처음 저장할때는 '자료1.ppt'이라고 저장을 했다가 계속 수정을 거치면서

'자료1_최종.ppt' , '자료1_진짜_최종.ppt' , '자료1_진짜_진짜_진짜_최종.ppt' 이렇게 변질되어가는 파일을

본 경험이 있쥬?ㅋㅋㅋ wow

이 때 이 파일들을 복사,백업,저장 등을 하는 행위들을 !!!!!!!!!!!버전 관리!!!!!!!!!!!라고 합니다 ㅋㅋㅋ

깃, 깃허브를 사용하면 버전관리 보다 좀 더 포괄적인 학술 분야의 형태로 넓히는 근간을 이야기한다고도 합니다.

Git이란 ? ? ?

좀 더 예를 들어 볼까요 ?

한글 문서작업을 팀원들이 동시에 작업 한다면 이 문서는 어떻게 관리 되어야 될까요 ?

팀원 10명이 하나의 파일에서 작업을 진행한다면 누군가는 변경된 글들을 하나하나 다 합치는 행위도 할 것이고

하루가 다르게 업데이트가 될 것인데 매일 또는 매 시간 단위로 업데이트되는 파일을 팀원과

어떻게 공유할 것인가요? 또 누군가의 실수로 이제껏 했던 작업물 내용이 엉뚱한 내용으로 바껴있거나

삭제되었을 때는 !!? 또 누가 어떤 작업을 했는지 손 쉽게 보고싶다면 !? 등등 생각해야할게 너무 많죠 ?

이런 문제점들을 도와주는 친구가 바로 Git이라는 친구입니다 .!!!!

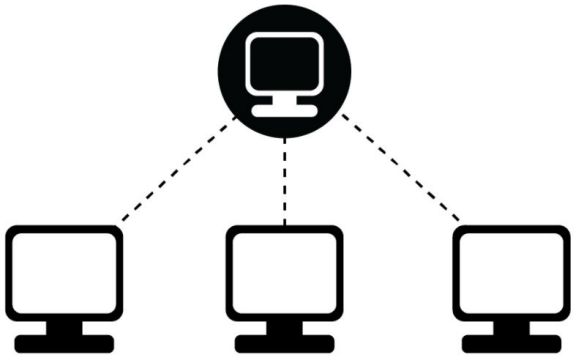
SVN이란 ? ? ?

버전 관리

1. 클라이언트 - 서버 모델

   하나의 중앙서버가 존재, 중앙 서버에서 각자 맡은 파트만 가져와서 작업, 다시 중앙으로 통합

   대표적 시스템 : CVS, SVN 등이 있다.



Git이란 ? ? ?

버전 관리

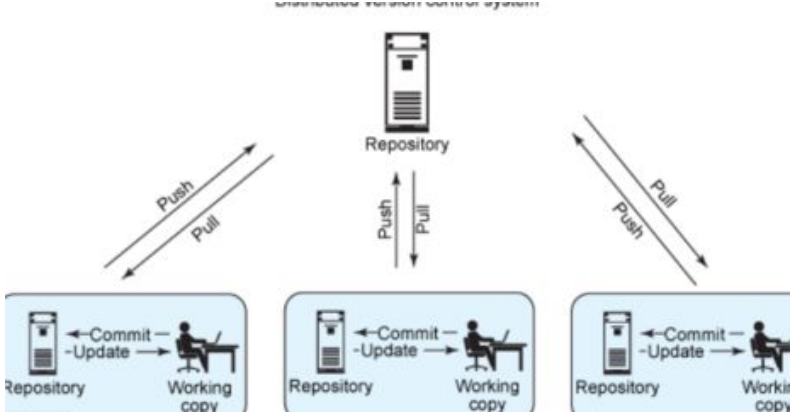
2. 분산 모델

하나의 중앙 서버가 존재하지만, 여러 클라이언트들은 각자의 컴퓨터 자장소(로컬)에 중앙 서버의 전체 사본을 가지고 있고 작업을 진행하는 것

대표적인 프로그램 : Git

여기서 어려운 말로 Git을 분산 버전 관리 시스템이라고 표현하였는데요.

분산 버전 관리 시스템을 쉽게 말하면, 여러명의 개발자(분산)가 특정 프로젝트를 자신의 컴퓨터로 협업하여 개발하면서 버전을 관리할 수 있는 시스템이다



Git이란 ? ? ?
SVN(클라이언트 - 서버)과 Git(분산 모델)의 차이점

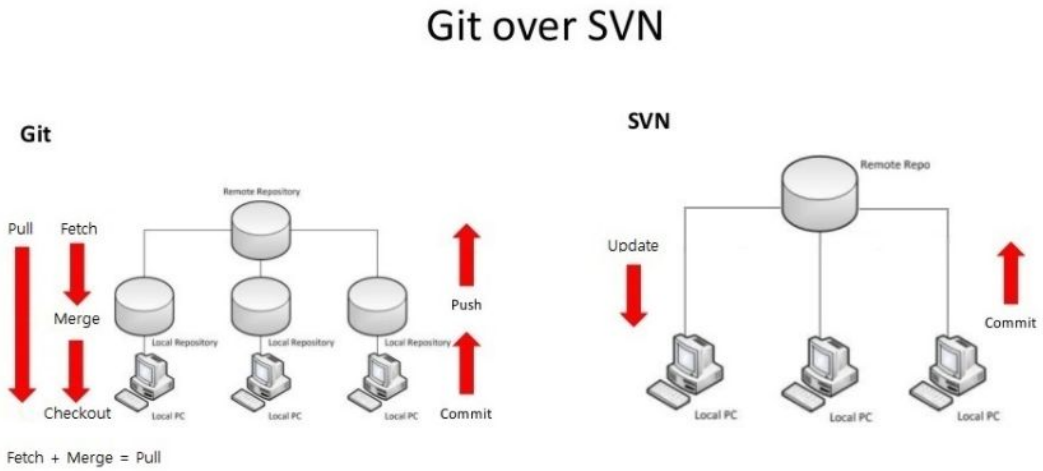
SVN : 중앙 서버에 소스 코드와 히스토리를 저장 (중앙 서버 <-> 클라이언트)

Git : 소스 코드를 여러 개발 PC와 저장소에 분산해서 저장 그렇기 때문에 어떠한 이유로 중앙 서버에 문제가 생기더라도

로컬 저장소에 커밋 가능 ! , 로컬 저장소들을 이용하여 중앙 저장소의 복원도 가능!

사본을 로컬에서 관리하기 때문에 Git이 SVN에

비해 훨씬 빠르다고 합니다.



Git VS SVN
SVN(클라이언트 - 서버)과 Git(분산 모델)의 차이점

SVN이란 ?

SVN 내 로컬PC에서 Commit을 하면 바로 중앙 저장소(Server)에 반영이 됩니다.

반면에 Git은 내 로컬 PC에서 (Commit)을 하면 로컬 저장소에 반영이 되고 로컬 저장소에서 ***Push***를 해야지만

원격 저장소(Server)에 반영이 됩니다!!

SVN 방식의 장점은 직관적이죠 왜냐하면 모든 사람이 중앙서버에 있는 같은 자료를 받아오고 Commit을 하는 순간

모든사람에게 공유가 된다 (이러한 방식의 단점은 만약 두 사람이 하나의 파일을 동시에 수정하고 커밋하였을 때 충돌이 일어날 확률이 높다!)

Git VS SVN
SVN(클라이언트 - 서버)과 Git(분산 모델)의 차이점

Git이란 ?

Git은 SVN에 비하여 직관적이지 못하고 적응하는데 시간이 좀 필요하다

한명이 작업한 작업물을 원격저장소에 올리려면 로컬 PC에서 Commit을 하고 로컬 저장소(개인PC)에 반영 후

원격저장소에서 한명이 작업후 올려준 결과물을 fetch로 자신의 로컬저장소로 마스터파일을 받아와서(깃 허브라면 서버에서 바로 작업 가능)

충돌이 나지않게 merge를 이용하여 병합을 하고(합치고)

로컬 저장소의 내용을 한번 더 Push(깃 허브 사용시 생략 가능)하여 원격저장소에 올리고 다른 사람이 작업 내용을 공유받을 수 있어요

즉 몇가지 과정이 추가되는거죠 !! 하지만 Git의 장점은 모든 작업이 로컬에서 이루어지기때문에 개발 시 처리속도가 빠르고 (네트워크 사용을 많이 하지않음)

웹 상에 저장소를 둘 수 있기 때문에 언제 어디서나 협업을 할 수 있죠! 그리고 중앙 서버에 에러가 생기면 모든 작업이 마비되는 SVN과 다르게

각 자의 로컬 저장소(개인PC)에 파일들이 저장되어있으므로 에러 발생시 복구도 용이하고 히스토리 관리도 용이합니다 !!

Git의 필요성

협업을 하는 경우에 같은 페이지를 업데이트하는 경우가 생길 수 있는데

내가 무언가를 변경하고 저장하고 , 다른 동료가 동시에 내가 작업한 페이지에서 작업을 한 후 저장을 하게되면

코드가 덮어 씌어지거나 했던 부분이 없어져 버릴 수 있는 위험이 생기겠죠?

하지만 **Git을 이용하면 이런일을 사전에 방지**해 줄 수 있습니다..

중복되지 않은 부분은 Git이 코드를 합쳐주고 행여나 같은 곳을 수정하여 중복(충돌)이 발생하게되면

사전에 알림으로 충돌이 일어났다는 것을 알려줌으로써 사전에 대응이 가능합니다. (또 매콤하지만 충돌 난 부분을 살릴 수도 있습니다 feat. 수작업)

Git의 장점

1.**소스 코드를 주고 받을 필요 없이**, 같은 파일을 **여러 명이 동시에 작업하는 병렬 개발이 가능**하다.

(브랜치를 통해 개발한 뒤, 본 프로그램에서 합치는 방식(Merge)으로 개발을 진행할 수 있다.)

2.**분산 버전 관리** 이기 때문에 인터넷이 연결되지 않은 곳에서도 개발 진행가능, 중앙 저장소가 문제가 생겨도 원상복구 가능.

3.팀 프로젝트가 아닌, 개인 프로젝트일지라도 Git을 통해 버전 관리를 하면 체계적인 개발이 가능하고, 문제 시 복구가 수월하며, 프로그램이나 패치를 배포하는 과정도

간단해진다.

Git 과 Github의 관계

일을 하다보면 협업하고 있는 코드를 저장할 서버가 필요하게 됩니다.

이때 버전 관리 시스템을 지원하는 웹호스팅 서비스(GitHub)의 기능을 통해

push(작업한 코드 올리기),

pull(팀원이 작업한 코드 가져오기),

request(병합(코드 합치기) 요청 pullRequest)와 같은 이벤트에 반응하여 자동으로 작업을

실행하게 할 수 있습니다.

Git 기본용어

Git을 사용하기 위해 알아야 할 기본 용어 몇가지입니다 .

Repository: 저장소를 의미하며, 저장소는 히스토리, 태그, 소스의 가지치기 혹은 branch에 따라 버전을 저장한다. 저장소를 통해 팀원들이 변경한 모든 히스토리를 확인 가능!

Working Tree : 저장소의 어느 한 시점을 바라보는 작업자의 현재 시점.

Stagin Area : 저장에 커밋하기 전에 커밋을 준비하는 위치.

Commit : 현재 변경된 작업 상태를 점검을 마치면 확정하고 저장소에 저장하는 작업.

Head : 현재 작업중인 Branch를 가리킨다.

A 그림(Head 1), B 그림(Head 2), C 그림(Head 3)등… 여러 그림이 있다고 생각하면 됩니다.

Branch : 가지 또는 분기점을 의미하며, 작업을 할때에 현재 상태를 복사하여 Branch에서 작업을 한 후에 완전하다 싶을 때 Merge를 하여 작업 함.

-> A그림을 그리다가 색감을 완전히 바꾸고싶어서 시도를 해보려고 하는데 A그림에 바로 시도를 했는데 만약 기존 그림과 어울리지 않는다면 여태껏 그린 그림은 날라가겠죠?? 이러한 일을 방지하는 방법은 복사기로 그림을 한장 복사한 후 원본 그림은 잘 보관해두고 복사한 사본 그림에 얼마든지 새로운 시도를 해볼 수 있겠죠? 브런치도 그런 개념입니다.! 새로운 복사본그림(브런치)를 하나 출력해서 마음껏 시도해보세여 원본은 잘 보관되어지고있으니까요! ㅋㅋㅋ

Merge : 다른 Branch의 내용을 현재 Branch로 가져와 합치는 작업을 의미한다.

Git 주요 명령어

git init : 깃 저장소를 초기화한다.(깃 저장소로 사용하겠다는 선언하는 명령어). 저장소나 디렉토리 안에서 이 명령을 실행하기 전까지는 그냥 일반 폴더입니다. git init으로 초기화를 한 이후에 추가적인 Git 명령어or Git명령어를 이용해 사용할 수 있다.

git help : 명령어를 알려주는 커맨드

git status : 저장소 상태를 체크한다. 어떤 파일이 저장소 안에 있는지, 변경 사항이 있는지, 커밋이 되어있는 상태인지 현재 저장소의 어떤 브랜치에서 작업하고 있는지 등을 볼 수 있다.

git clone : 원격 저장소의 저장소를 내 local에서 이용할 수 있게 download받는다고 생각하면 된다.

git add : 이 명령이 저장소에 새 파일들을 추가하진 않지만 대신 깃이 파일들을 지켜보게하여 변동사항이 일어났는지 파악할 수 있다. 파일을 추가하면, 깃의 저장소 “스냅샷”에 포함된다.

git commit : 깃의 의미있는 수정 작업이 끝났을 때 마침을 알리는 작업이다. **ex) “git commit -m “Message” -m** 명령어의 “**Message**”부분은 해당 커밋작업에 대한 ‘주석’을 남기는 부분이다 어떠한 작업을 했는지 욕을 제외한 **남기고 싶은 주석 아무거나 남기면 된다**.

git push : 로컬 컴퓨터에서 작업하고 커밋을 깃허브에서 온라인 서버에 올리는 작업이다 “push”를 사용해서 올리면된다. (이 후 실습단계에서 더 자세히 다루겠습니다)

Git 주요 명령어

git pull : 로컬 컴퓨터에서 작업할 때, 저장소의 변경된 내용을 로컬(내 컴퓨터)저장소로 다운+업데이트 하는 작업입니다.

예를들어 현재 내 파일에는 “안녕하세요”라고 적혀있는데 다른 동료가 그 파일에 내용을 추가했다면 pull을 받고 나면

“안녕하세요 메테오입니다.” 라고 파일의 내용이 변경되게 됩니다.

git log : 저장소에서 내 커밋 내역을 확인해보고 싶을 때 사용하는 명령어 입니다.

git branch : 새로운 작업을 현 시점의 파일에 영향을 받지 않고 변경, 삭제 등 테스트를 원한다면 이 명령어로 새로운 브랜치를 만들고 독립적인 공간을 만들어 주면 됩니다.

git checkout : 독립된 작업 공간인 branch를 이동할 수 있는 명령어이다 “ **git checkout ‘브랜치명’** ” 이런식으로 명령어를 사용하면 됩니다.

git merge : 독립적으로 생성되어진 브런치에서 테스트를 다 끝내고 돌아와 본 프로그램에 적용 시킬 때 병합하는 명령어입니다.

A branch(메인 브런치): “안녕하세요” | B branch: “hello” merge -> A branch : “hello”

"git merge hello"라고 입력한다면 hello브랜치에서 만든 모든 변경사항을 master로 추가한다.

실습 (Local & GitBash)

1. Git Bash 설치하기 (https://git-scm.com/) 이동 해주세요~

2. Git Bash를 다운 (참고 사이트 : https://gabii.tistory.com/entry/Git-Git-Bash-219-%EC%84%A4%EC%B9%98%ED%95%98%EA%B8%B0)

3. 다 되었으면 숨 한번 크게 들이쉬고 가겠습니다. 쉬우니까 천천히 잘 따라오시면 됩니다. 긴장하지 마세요

   어렵다 생각한 순간부터 어려워지고 쉽다고 생각한 순간부터 쉬워집니다!! :)

실습 (Local & GitBash)

설치는 다 하셨나요 ? 그러면 우선 깃의 개념이 꼭 코딩에만 국한 되는것은 아니니 간단한 테스트를 통해 위에있던 명령어,

깃 설정, 깃 어드 푸쉬 등을 한번 직접 실습해보도록 하겠습니다.! 레쓰 기릿

1. 하기에 앞서 간단하게 예시를 들어 정리를 한번 더 하고 실습하겠습니다

   (참고 출처:데어프로그래밍 블로그프로젝트 Git 강의
   링크 :https://www.youtube.com/watch?v=9Nk1a6UMAqo&list=PL93mKxaRDidECgjOBjPgI3Dyo8ka6Ilqm&index=8)

   아름다운 풍경을 보고있습니다. 너무 아름다워서 사진을 찍었어요 ! 그리고 오래오래 간직하고싶어서 사진첩에 담아두었어요

   여행을 하는동안에 수많은 아름다운 풍경을 보았고 그 풍경들을 사진으로 찍어 사진첩에 많이 저장했네요 !

   그러면 앞으로 이 시간들이 생각나면 사진첩에서 이 풍경들을 꺼내서 사진을 보곤하겠죠

- ***어떠한 풍경을 사진을 찍는다 -> 장면을 기록한다 -> (스냅샷 : .add 명령어)***
- ***이 사진을 사진첩에 담는다 -> 오래오래 보관한다 -> (헤드 영역에 저장 : commit 명령어)***
- ***깃 3가지 영역 작업영역 , 인덱스 , 헤드 이 3가지 영역을 일치시킨다 -> 동기화 한다.***

실습 (Local & GitBash)

1. 우선 바탕화면에 아무 폴더를 만들어주세요 (이름은 자유롭게 지으셔도 됩니다)

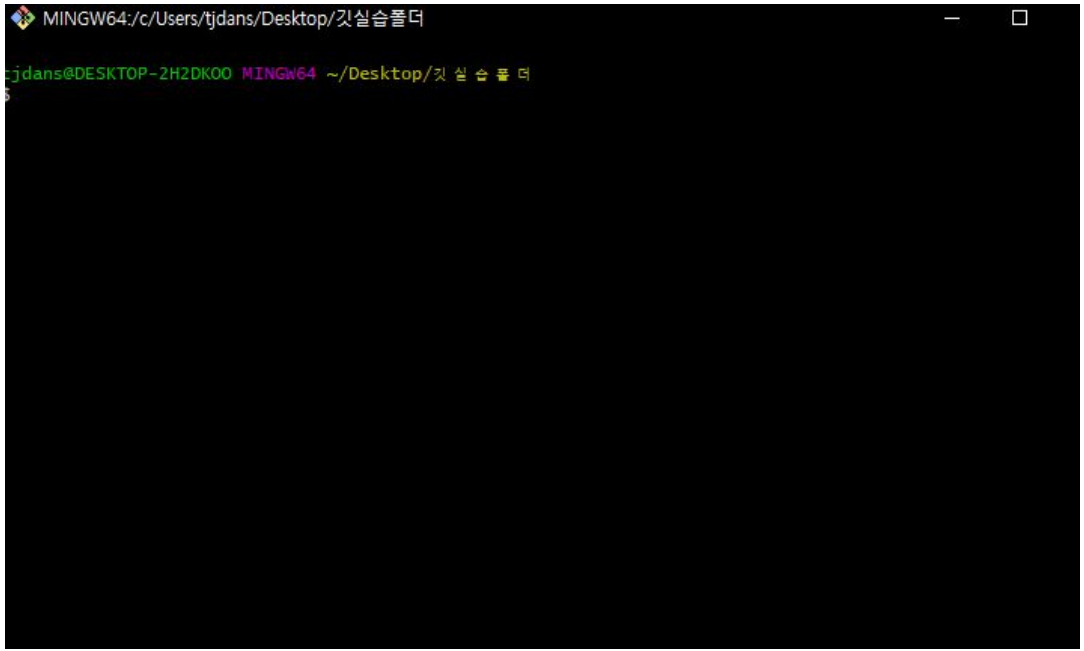
실습 (Local & GitBash)

2. 그다음 새로운 텍스트 파일을 하나 만들어 주도록 하겠습니다.

실습 (Local & GitBash)

3. 그리고 이전에 다운받았던 깃 배쉬를 실행해볼게요 현재 폴더에서 우클릭을 해줍니다

4. 그리고 Git Bash Here을 눌러주세요 아래와 같은 창이 하나뜨죠 ?



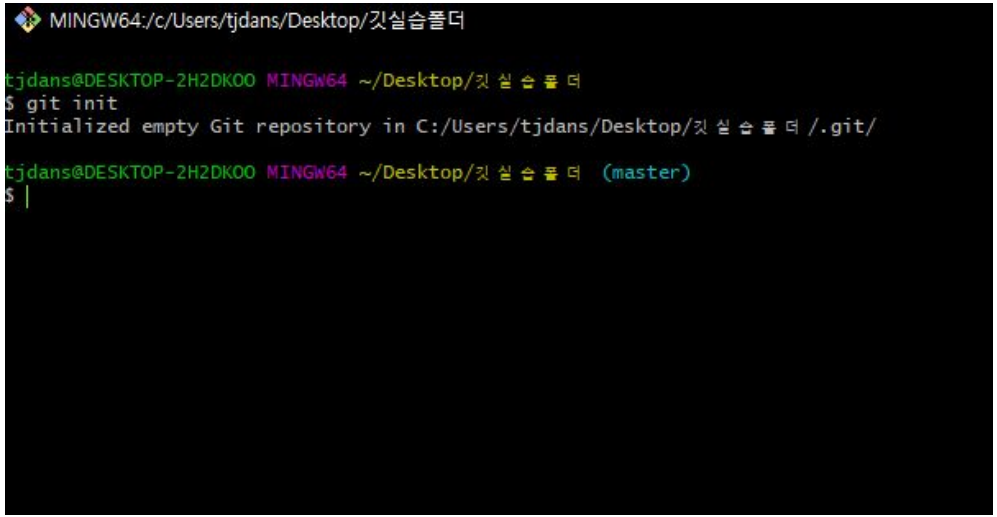
실습 (Local & GitBash)

5 명령어를 실행 해보겠습니다 git init이라고 명령어를 입력후 엔터를 쳐주세요 아주아주 과감하게 말이죠

명령어를 실행 하고 나니 뭔가가 블라블라 나오고 갑자기 (master)라는 글과 못보던 .git이라는 폴더가 ㅅㅡ 생겼습니다 .

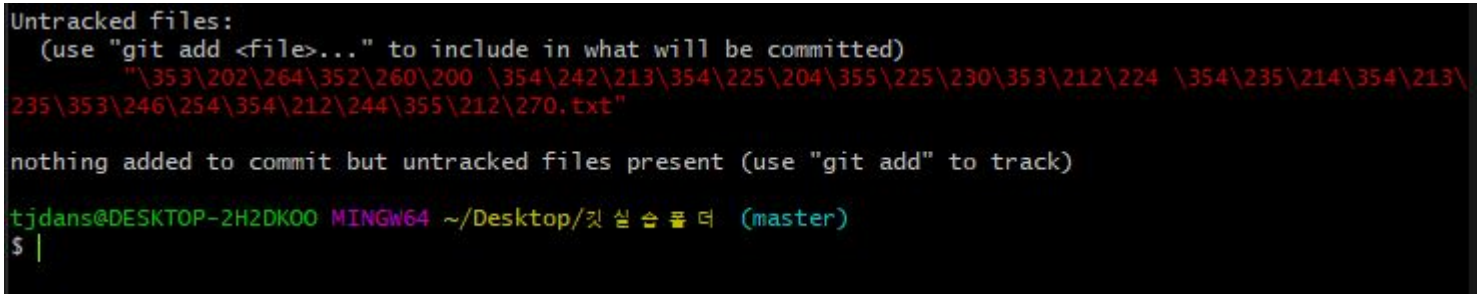
이제 하나의 폴더(작업 영역)이 생긴거죠

이전 자료에 적혀 있듯이 git init 명령어를 만나기전에는 단순한 폴더였던 친구가 이제는 git에 의해 관리되어지는 폴더로 변경된것이죠 ㅋ



실습 (Local & GitBash)

6. 자 git status를 입력 후 엔터를 눌러볼까요 ?



Untracked files : 블라블라 뭔가 시뻘겋게적힌글이있습니다 그런데 뒤에 확장자를 보니 .txt라고 적혀있네요 맞아요 이건 바로 우리가

만들었던 파일(내가 좋아하는 음식리스트)에요 빨간이유는 즉 **Git이 이 폴더를 감시(작업 영역)**하고있다는거에요 ㅋㅋ

야 니폴더에 뭐하나 생겼더라 ? 어쩔건데 그래서

사진찍을거야 말거야~~~ 알려주는거죠 두유노 왓아임쎄잉 ?

실습 (Local & GitBash)

7. 그러면 이 폴더에서 생성 된 파일을 **스냅샷(SnapShot)을 찍어 Index영역**으로 이 데이터를 넘겨보겠습니다 ! 즉 사진을 찍어볼까요?

자 자 웃으세요 스마일~ 😁 git add . 명령어를 입력 후 엔터~!

add 이후 **띄우고 .을 입력**해주세요 보통 .을 찍어요 **.은 이 폴더안의 모든 파일**을 입력하겠다 이런의미입니다 쿼리문에서 *하고 비슷해요~!

그러면 뭔가 슥 하고 지나가고 아무런 정보도 출력되지 않네요 o_o?

실습 (Local & GitBash)

8. 하지만 **git status(현재 작업영역의 상태등을 보는 명령어)** 명령어를 다시 실행해볼까요 ?

```
tjdans@DESKTOP-2H2DKOO MINGW64 ~/Desktop/깃 실 습 폴 더  (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   "\353\202\264\352\260\200 \354\242\213\354\225\204\355\225\230\353\212\224 \354\235\
214\354\213\235\353\246\254\354\212\244\355\212\270.txt"


tjdans@DESKTOP-2H2DKOO MINGW64 ~/Desktop/깃 실 습 폴 더  (master)
$
```

오 ? 빨간색이었던 글들이 초록색으로 변경이 되었네요 이 말은 add명령어로 사진을 찍었고 이제 인덱스 영역에 해당 폴더의 내용이 들어왔단 거에요 즉 **작업영역과 인덱스 영역까지 동기화가 되었단 말이죠 !**

실습 (Local & GitBash)

9. 인덱스 영역까지 동기화가 되었다 즉, 스냅샷을 찍었다 이 말이죠 이 말이 무슨 말인지 감이 안잡히시죠 ? 그러면 새로운 파일을 하나 더 만들어보고 git status 명령어를 다시 실행시켜볼까요 ?

그러니 파일 관련 된 글이 두 줄이 되었습니다.

한 줄은 초록색

한 줄은 빨간색

초록색 관련 글은 아까 우리가 add를 시켰기 때문에

인덱스 영역에 잘 동기화되었어~~~ 이말 이죠

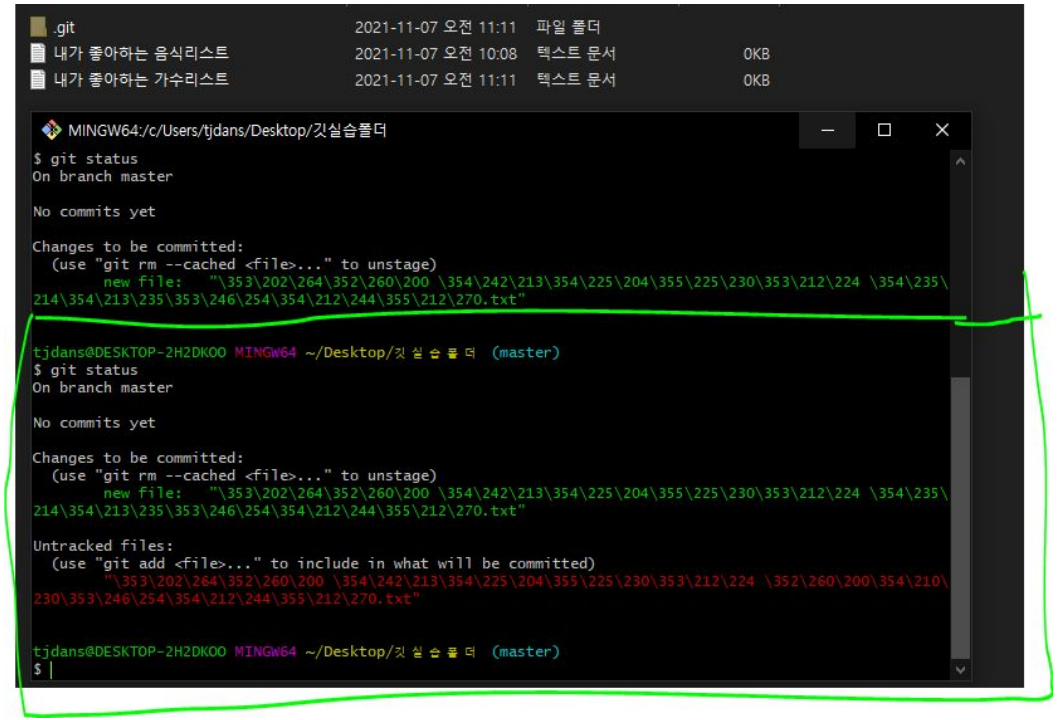
그런데 새로만든 파일은 “야 너 또 파일만들었더라 ?

저장 시킬거야 말거야~~~” 이렇게 또 물어보고 있는

상황인거죠 ㅋㅋㅋ 새로 만든파일을 삭제하고

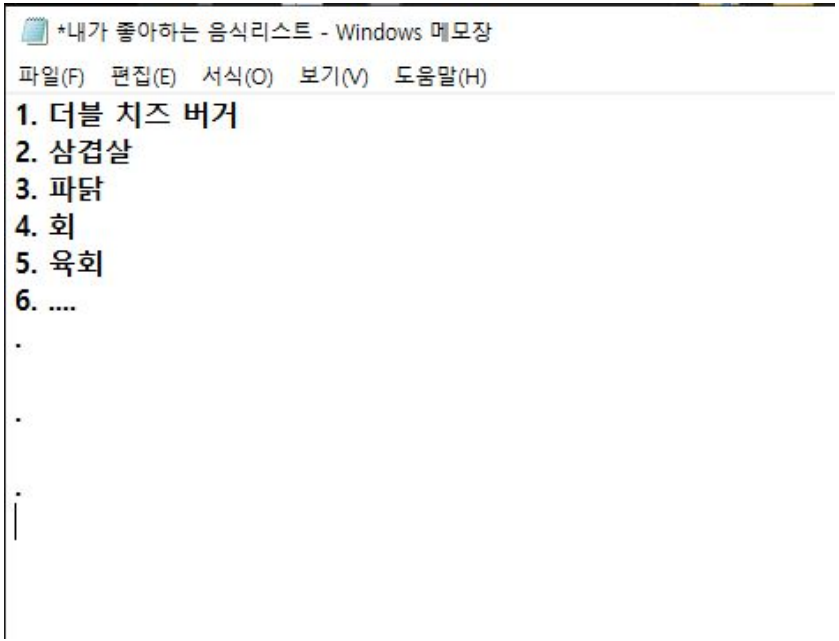
git status 명령어를 다시 실행해보세요 그러면 이해가 더

가실거에요 ! ㅋㅋ



실습 (Local & GitBash)

10. 자 이제 내가좋아하는 음식리스트에 텍스트를 작성해보도록 하죠



실습 (Local & GitBash)

11. 그리고 다시 깃배쉬 cmd창으로가서 git status명령어를 실행시켜주세요

그러면 초록글과 빨간 글이 하나씩 뜰 거에요

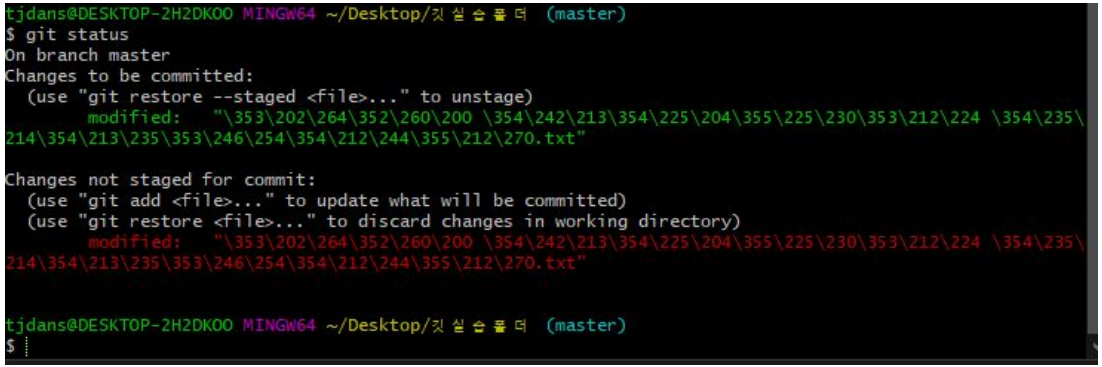
이런식으로 말이죠

즉 아까 commit을 날려 있는 상태와

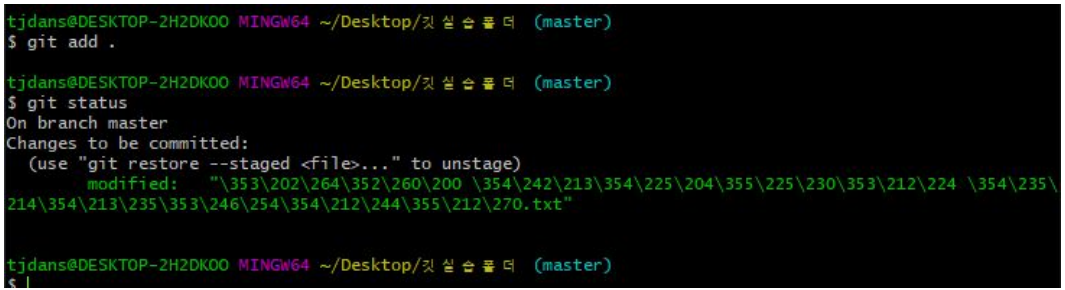
우리가 좋아하는 음식들을 작성하면서

변동된 데이터가 감지되어 git이 알려주는거죠

“지금 파일에 변동사항있으니까 빨리 저장해!”



이렇게 말이죠 그러면 git add . 명령어를 날리고 git status를 다시 실행 시켜보겠습니다.



실습 (Local & GitBash)

12. 그러면 다시 초록색 글로 바뀌었어요 즉 니가 변동한 사항 사진(SnapShot) 찍었어~ 그래서 인덱스영역에 내가 저장을 했어

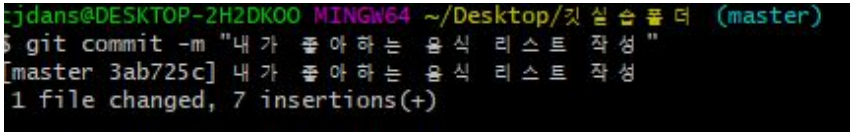
    이 말이되는거죠 앞에 실습처럼 git init을 하는 순간 그 폴더를 git에게 “작업 영역”으로 만들면 변동되는 것들을 감지하고 그 변동된

    데이터들을 스냅샷을 찍을 수 있죠 딱 여기까지 실습한거에요 ! 그러면 이제 찍은 사진을 사진첩으로 저장하는 것을 해볼게요 !

    즉 오래오래 데이터를 보관할 수 있게 사진첩에 저장할게 ~~ 이 말이죠 시작해볼게요 레쓰기릿

실습 (Local & GitBash)

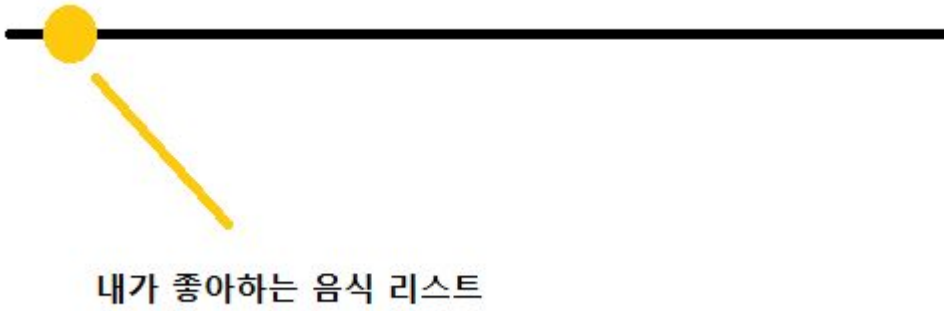
13. git 실행창에 git commit -m “남길 메시지” 이렇게 명령어를 실행 시켜볼까요 ? -m은 현재 사진첩에 저장될 때 저장되어지는 “파일명”이라고 생각하시면될거에요 정확히는 파일명보다는 사진(파일)에대한 설명(주석)이죠 ㅇㅇ에서 본 사진 이런식으로 말이죠 해봅시다 바로 -m “남길 메시지(주석)”은 항상 습관처럼 해주세요 좋은 습관을 들이자구요



그러면 어떤 알림 메시지가 뜨죠 ! 즉 사진첩에 파일을 저장했다 이 말이에요

git으로 설명하자면 브런치의 하나의 헤드 영역이 생성된 것이죠 !!

아래 그림처럼 말이죠

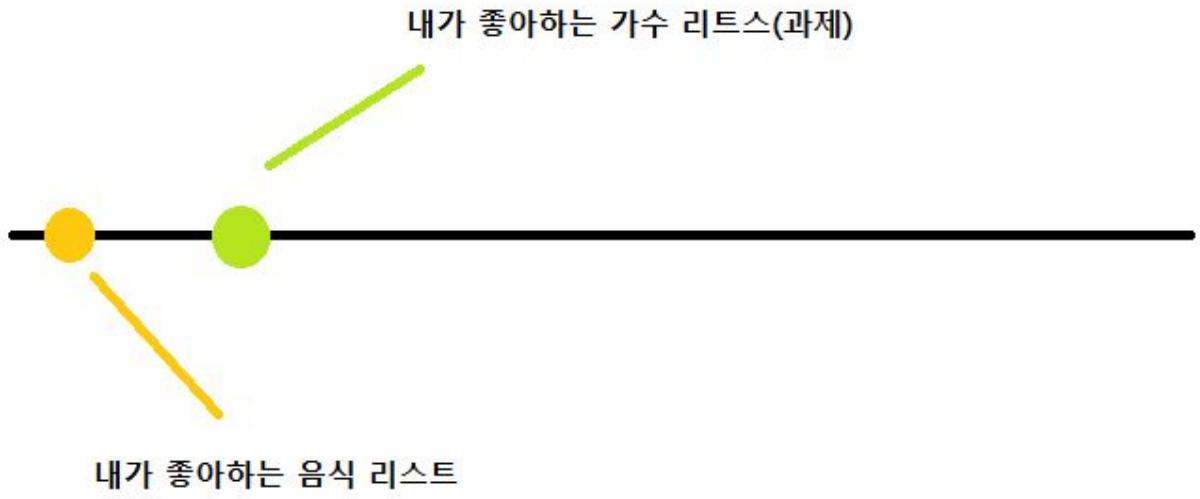


실습 (Local & GitBash)

14. 자 과제 입니다 새로운 파일 “내가 좋아하는 가수리스트”파일에 좋아하는 가수를 적고 이 브랜치 영역까지 (사진첩에 저장 , commit) 완료해주세요. * 브랜치영역 즉 헤드를 만드는 순간 < 작업영역 | 인덱스 영역 | 헤드 영역 > 이 다 동기화 되었단 거에요 !

꼭 과제 실습을 해주세요 로컬 * 깃배쉬 연습은 여기까지 하겠습니다. 물론 이 외적으로 깃 브랜치 생성 병합 등 많은 것들이 있어요

하지만 이 기초중의 기초만해도 깃을 사용하는데 무리는 없으니까요 ! **깃 브랜치 생성 병합등은 다음 업데이트 버젼으로 찾아뵙겠습니다.**



Thank You!