Git

Git 이란?

- **Git**은 형상 관리 도구 중 하나로, 컴퓨터 파일의 변경사항을 추적하고 여러 명의 사용자들 간에 해당 파일들의 작업을 조율하기 위한 분산 버전 관리 시스템이다.

- **GIT**은 형상 관리 프로그램

- 이전에는 SVN으로 형상 관리를 했지만 요즘은 GIT으로 형상 관리를 하는 것이 추세이다

- SVN은 변경 사항이 없는 파일도 확인을 하지만 GIT은 변경 사항이 있는 파일만 확인을 하므로 속도면에서 우세하기 때문이다

GitHub

- **깃허브(GitHub)**는 분산 버전 관리 툴인 깃(Git)을 사용하는 프로젝트를 지원하는 웹호스팅 서비스이다.

- Ruby on Rails로 작성되었다.

- GitHub는 영리적인 서비스와 오픈소스를 위한 무상 서비스를 모두 제공한다.

Git 설치

https://git-scm.com

window 버전 다운로드 – 모두 default 설치

- 실행 프로그램

=> Git Bash : 리눅스 환경

=> Git CMD : 윈도우 커맨드 창

=> Git GUI : 윈도우 환경

- Git Bash를 실행한다.

yeoni @ LAPTOP-5NQVQFFF MINGW64 ~

$

컴퓨터_사용자_이름 @ 컴퓨터_이름 MINGW32 현재_폴더_위치

- MINGW32는 프로그램 이름이고, $는 입력커서이다.

- 설치 버전 확인

yeoni@LAPTOP-5NQVQFFF MINGW64 ~

$ git --version

git version 2.29.2.windows.3

리눅스 기본 명령어

① cd (Change Directory)

② ls (리스트 확인 가능) => ls -l 또는 ll 쓰면됨(list의 l임) ,

ls -arlth (권한,숨김파일 시간순 정렬)

③ cat (현재 파일의 내용을 확인할 수 있음)

④ touch (파일 만들기)

⑤ mkdir (make directory 디렉토리 만들기)

⑥ clear (현재 창 깨끗이 지우기)

⑦ rm -rf (파일, 디렉토리 삭제)

⑧ cd .. (윗 경로로 이동)

⑨ cp 파일 ./디렉토리 (복사)

⑩ vi 파일 (파일에서 작업 가능) => i 누르면 파일에 데이터 입력 가능 esc누르면복귀

⇨ :q! => 저장 안하고 나가기

⇨ :wq, :wq! => 저장하고 나가기

Git 환경 설정

- git에서 커밋 할 때마다 기록하는 사용자 이름과 메일 주소를 설정해야 한다.

- 사용자 계정 홈 디렉토리에 .gitconfig 파일이 생성된다.

  C:₩Users₩yeoni₩.gitconfig

- 리눅스 기본 명령어로 실습

```
rlawo@KJW MINGW64 ~
$ git config --global user.name "UsoD98"

rlawo@KJW MINGW64 ~
$ git config --global user.email “rlawodnjs07@gmail.com”

rlawo@KJW MINGW64 ~
$ git config -list

rlawo@KJW MINGW64 ~
$ cd d:

rlawo@KJW MINGW64 /d
$ cd Git

rlawo@KJW MINGW64 /d/Git
$ ls
lib/  workspace/

rlawo@KJW MINGW64 /d/git
$ ls -l
total 0
drwxr-xr-x 1 rlawo 197609 0 Jan 12 11:20 lib/
drwxr-xr-x 1 rlawo 197609 0 Jan 12 11:21 workspace/
(42142 14 2)

rlawo@KJW MINGW64 /d/git
$ cd workspace/

rlawo@KJW MINGW64 /d/git/workspace
$ ls -l
total 0
```

(워크스페이스에 아무것도 없어서 이런 결과가 뜬다)

```
rlawo@KJW MINGW64 /d/git/workspace
$ ls -l
total 1
-rw-r--r-- 1 rlawo 197609 54 Jan 12 11:32 fruit.txt
```

```
rlawo@KJW MINGW64 /d/git/workspace
$ cat fruit.txt
apple
banana
melon
apple
tomato
strawberry
grape

rlawo@KJW MINGW64 /d/git/workspace
$ touch animal.txt    => 새 파일 만들기 가능

rlawo@KJW MINGW64 /d/git/workspace
$ ls -l
total 1
-rw-r--r-- 1 rlawo 197609  0 Jan 12 11:48 animal.txt
-rw-r--r-- 1 rlawo 197609 54 Jan 12 11:32 fruit.txt

*디렉토리에 angel 폴더 생성 후
rlawo@KJW MINGW64 /d/git/workspace
$ ls -l
total 1
drwxr-xr-x 1 rlawo 197609  0 Jan 12 11:49 angel/
-rw-r--r-- 1 rlawo 197609  0 Jan 12 11:48 animal.txt
-rw-r--r-- 1 rlawo 197609 54 Jan 12 11:32 fruit.txt

rlawo@KJW MINGW64 /d/git/workspace
$ mkdir devil

rlawo@KJW MINGW64 /d/git/workspace
$ ls -l
total 1
drwxr-xr-x 1 rlawo 197609  0 Jan 12 11:49 angel/
-rw-r--r-- 1 rlawo 197609  0 Jan 12 11:48 animal.txt
drwxr-xr-x 1 rlawo 197609  0 Jan 12 11:50 devil/
-rw-r--r-- 1 rlawo 197609 54 Jan 12 11:32 fruit.txt

rlawo@KJW MINGW64 /d/git/workspace
$ rm -rf animal.txt

rlawo@KJW MINGW64 /d/git/workspace
$ ls -l
total 1
drwxr-xr-x 1 rlawo 197609  0 Jan 12 11:49 angel/
drwxr-xr-x 1 rlawo 197609  0 Jan 12 11:50 devil/
-rw-r--r-- 1 rlawo 197609 54 Jan 12 11:32 fruit.txt

rlawo@KJW MINGW64 /d/git/workspace
$ rm -rf devil         => devil(/ 생략 가능)
rlawo@KJW MINGW64 /d/git/workspace
$ ls -l
total 1
drwxr-xr-x 1 rlawo 197609  0 Jan 12 11:49 angel/
-rw-r--r-- 1 rlawo 197609 54 Jan 12 11:32 fruit.txt
```

```
rlawo@KJW MINGW64 /d/git/workspace
$ cd ..          => 윗 경로로 이동

rlawo@KJW MINGW64 /d/git


rlawo@KJW MINGW64 /d/git/workspace
$ cp fruit.txt ./angel      => angel 폴더에 파일 복사
rlawo@KJW MINGW64 /d/git/workspace
$ ls -l
total 1
drwxr-xr-x 1 rlawo 197609  0 Jan 12 11:56 angel/
-rw-r--r-- 1 rlawo 197609 54 Jan 12 11:32 fruit.txt

rlawo@KJW MINGW64 /d/git/workspace
$ vi fruit.txt
(I 누르고 사과, 딸기 입력 후 esc => :wq!(느낌표 생략 가능)
rlawo@KJW MINGW64 /d/git/workspace
$ cat fruit.txt
apple
banana
melon
apple
tomato
strawberry
grape
사과
딸기
```

Git 저장소

- 저장소는 관리하고자 하는 모든 소스 코드 및 디렉토리가 저장되는 곳으로 여러 개의 프로젝트를 하나의 저장소에서 관리할 수 있다.

- 저장소는 특별한 데이터베이스처럼 보이지만 사실은 디렉토리와 파일 기반으로 데이터를 관리하며 생성한 저장소 하위의 .git 디렉토리에 저장되어 있다.

(1) Git 저장소 기반으로 프로젝트 관리하는 방법

- Git 저장소에 신규 프로젝트를 생성해서 반영하는 방법으로 처음으로 프로젝트와 저장소를 생성할 때 사용한다.

- Git 저장소에 누군가가 이미 만들어 놓은 프로젝트를 활용하고 싶다면 해당 프로젝트를 복제해서 사용할 수 있다.

(2) Git 저장소 생성

- Git 저장소로 사용할 디렉토리를 생성한다.

=> D:₩git_home₩git_repo 디렉토리 만들기

=> 윈도우 탐색기에 폴더로 작성해도 되고 리눅스로 만들어 된다.

- git init

=> 생성한 디렉토리를 Git이 저장소로 인식할 수 있도록 초기화한다.

=> 초기화하면 .git 디렉토리가 생성된다.

=> "." 이 붙어서 숨김 디렉토리로 인식한다.

=> 해제하려면 .git를 삭제하면 된다.

Git으로 형상 관리하기

(1) 기본 용어

1) 스냅샷 (Snapshot)

- Git에서 커밋할 때마다 발생하며 커밋한 시점의 형상 관리 상태를 의미한다.

- 버전이라는 의미도 포함하고 있다.

2) 트리(tree)

- 파일과 디렉토리의 구조 정보를 저장하고 있다.

- 파일 시스템이 트리구조를 가지고 있기 때문에 형상 관리 역시 트리 형태로 스냅샷을 저장한다.

3) 저작자(Author)

- Git에서 관리하고 있는 파일 혹은 디렉토리를 최초로 생성한 사람의 정보이다.

- 일반적으로 사람을 식별할 수 있는 이름, 이메일 등의 정보를 저장한다.

4) 커미터(Committer) - 파일을 변경한 사람

- 최초 파일이 저장소에 반영되면 저작자와 커미터가 동일하지만, 이후 해당 파일을 다른 사람이 수정하면 커미터가 변경된다.

- 저작자는 파일을 생성한 사람, 커미터는 파일을 변경한 사람

5) 커밋 메시지(Commit Message)

- Git은 커밋할 때 반드시 커밋에 대한 메시지를 저장하도록 되어있다. - 명령어의 파라미터로 전달할 수도 있고, 에디터를 통해서 많은 내용의 메시지를 저장해서 전달할 수도 있다.

- 메시지는 되도록 상세히 작성해 두는 것이 좋다.

6) 부모 커밋(Parent Commit) - 현재 커밋이 참조하고 있는 상위 커밋을 의미한다.

- 최초 커밋시에는 부모 커밋 객체가 없지만, 이후 다시 커밋을 하게 되면, 현재 커밋된 객체가 부모 커밋 객체가 되고 나중에 커밋한 객체가 커런트 객체가 된다.

Git의 스테이징 단계

- git add으로 진행되는 형상 관리가 3가지 영역에서 진행된다.

① 워킹 디렉토리

- 소스 코드를 작업하는 영역으로 코드를 추가, 수정, 삭제하는 작업이 이루어지는 영역을 의미한다.

② 스테이징 영역

- 워킹 디렉토리에서 git add 명령을 실행하면 파일들은 Git의 스테이징 영역으로 이동하며 이를 통해 소스 코드의 상태 정보를 확인할 수 있다.

③ 저장소 영역

- 스테이징 영역에 있는 소스 코드에 git commit 명령을 실행하면 최종적으로 Git의 저장소에 반영된다.

파일 관점에서 Git의 4가지 상태

① **Untracked** : 워킹 디렉토리에 추가되었지만 Git에서 관리하지 않는 상태

② **Unmodified** : 신규로 파일이 추가되었을 때의 상태로 new file 상태와 동일

③ **Modified** : 파일이 추가된 이후 해당 파일이 수정되었을 때의 상태

④ **Staged** : Git의 스테이징 영역에 반영된 상태

스테이징에 있는 파일 삭제 – git rm --cached 파일명

Git 이력 조회 및 변경 내용 비교

① 이력 관리 기능

- 소스 코드가 언제 누구에 의해서 생성, 변경, 삭제를 했는 지 확인을 할 수 있다.

② 스냅샷

- Git에서는 커밋이 이루어질 때마다 모든 형상 관리 파일들을 바이너리 형태로 묶어서 관리하는 것

- Git에서 하나의 스냅샷은 하나의 커밋을 의미하며, 스냅샷 단위로 이력을 관리한다.

③ git log

- 이력 정보 확인

- 커밋한 작업자, 일시, 메시지를 확인할 수 있다.

  git log –p

  - 버전과 버전사이의 코드 상태에서의 차이점을 비교

  git log ––oneline

  - commit id의 일부와 commit message만 보여준다.

④ git diff

- 저장소의 파일과 워킹 디렉토리에 있는 파일을 비교해, 파일내의 콘텐츠 변경 부분을 확인할 수 있다.

브랜치와 머지

(1) 브랜치(Branch)

- 개발을 하다 보면 코드를 여러 개로 복사해야 하는 일이 자주 생긴다.
- 코드를 통째로 복사하고 나서 원래 코드와는 상관없이 독립적으로 개발을 진행할 수 있는데, 이렇게 독립적으로 개발하는 것이 브랜치다.
- 모든 버전 관리 시스템은 브랜치를 지원한다.

(2) 브랜치가 필요한 경우

- 동일한 상품이지만 고객사별 다른 대응(버전)이 필요할 때
- 개발 시, 일정기간 동안 실험적인 일(작업)을 하게 될 때
- 서비스 적용과 개발 진행을 병렬로 처리할 때

(3) 머지(Merge)

- 특정 규칙에 따라 순서대로 둘 이상의 파일을 합쳐 하나의 파일로 만드는 것

(4) 명령어

1) 브랜치 확인 – git branch
2) 브랜치 만들기 - git branch 브랜치명
3) 포인터 변경 - git checkout 브랜치명
4) 변경된 내용을 합치기 - git merge 병합할_브랜치명
   먼저 master로 포인터를 옮긴 후, master에 브랜치를 머지한다.
5) 브랜치명 삭제하기 - git branch -d 브랜치명

저장소를 만들게 되면 기본 브랜치는 'master' 브랜치 이다.

과거 특정 시점으로 돌아가는 방법

(1) reset

- 초기화 (취소)

- 되돌린 이후 버전 삭제

- 이미 push한 상태라면 reset은 할 수 없다

- reset를 했기 때문에 git log를 보면 버전을 사라졌지만

  수정된 파일의 내용은 그대로 남아있다. → 다시 git add / git commit 해야 한다.

(2) revert

- 되돌리기

- 되돌린 이후 버전 유지

- 버전 아이디를 역순으로 하나씩 되돌려야 한다.

- revert ~~ 라고 메시지가 들어간 버전이 하나 더 만들어지고

  수정한 파일의 내용은 사라지고 이전 버전의 파일 내용이 그대로 다시 나온다.

(3) 되돌리기

git reset --hard

git revert 버전아이디 (역순으로 해야 함)