리눅스 & git 보고서 작성

로빛 예비단원-소재웅

목차

리눅스

Git

- 1. Linux 란 ?

컴퓨터, 스마트폰, 가전제품 등 모든 전자 기기는 하드웨어와 소프트웨어로 구분할 수 있습니다. 하드웨어는 눈에 보이는 전자기기의 물리적인 모든 것을 의미하며, 소프트웨어는 특정 목적을 이루기 위해 컴퓨터에 내리는 명령들을 모아놓은 것을 의미합니다. 사용자가 소프트웨어(일반적인 애플리케이션으로서의 소프트웨어)를 사용할 때, 소프트웨어는 컴퓨터의 CPU, RAM 등의 하드웨어를 사용해서 사용자가 요구하는 동작을 수행하는데, 이때, 운영체제가 소프트웨어에 필요한 만큼의 하드웨어의 자원을 할당해 줍니다.

즉, 운영체제는 CPU, RAM과 같은 한정된 하드웨어 자원을 관리하고, 하드웨어와 소프트웨어 사이에서 이 둘을 중재해 주는 역할을 합니다. 여러분의 컴퓨터에 설치돼 있는 Windows, Mac OS, 그리고 스마트폰 및 태블릿에 설치돼 있는 Android, iOS 등도 모두 운영체제로, 생김새와 사용법은 조금씩 다르지만, 근본적으로 기기의 하드웨어 자원을 관리하는 역할을 수행합니다.

이번 콘텐츠의 주제인 리눅스도 운영체제의 일종입니다. 리눅스(Linux)는 핀란드의 소프트웨어 엔지니어 리누스 토르발즈(Linus Torvalds)가 유닉스(Unix)에 기반하여 만든 운영체제입니다. 리눅스는 1991년에 세상에 처음 등장한 이래로 현재까지 개인용 컴퓨터, 스마트폰, 자동차, 가전제품, 슈퍼컴퓨터 등 다양한 분야에 범용적으로 사용되고 있습니다.

[Linux 마스코트]

- 2. 리눅스를 사용하는 이유 3가지 이유

- 1. 오픈소스

-> 리눅스는 오픈 소스 운영 체제이다. 즉, 일반적으로 사용되는 window와 mac os와 달리 리눅스는 개인이든 법인이든 누구나 리눅스를 무료로 설치하여 사용할 수 있다.

- 2. 커스터마이징

-> 리눅스는 엄밀히 말하면 리눅스 커널(Linux Kernel)을 의미한다. 커널은 운영 체제의 핵심적인 기능을 수행하는 운영 체제의 한 부분이다. 즉, 리눅스는 커널의 형태로 만들어져 있어 운영 체제가 수행해야 하는 핵심 기능만 정의돼 있으며, 이외의 부분은 사용자가 자신의 용도에 맞게 커스터마이징 하여 사용할 수 있다.

- 3. 안정적인 동작

-> 현재까지도 리눅스는 오픈소스로 관리되고 있고 이에 따라 넓은 사용 범위에서 다양한 사용자들의 검증을 실시간으로 거치고 있으며, 위협 사례나 버그가 발견되면 불특정 다수의 사용자들에 의해 버그가 수정되어 왔기 때문에 다른 운영체제들보다 상대적으로 더 안정적인 동작을 할 수 있다.

- 3. 리눅스 기본 명령어

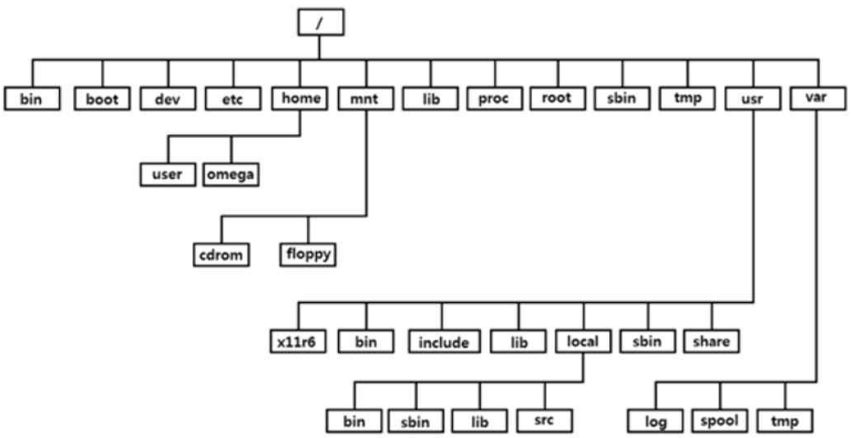
리눅스는 CLI(Command Line Interface)로 조작할 수 있다. 여기에서 Interface는 두 대상을 연결해 주는 어떤 매개체를 의미하며, CLI에서의 '두 대상'은 컴퓨터와 사람을 의미한다. 즉, CLI란 명령어(Command Line)를 통해 컴퓨터를 제어하는 방식을 의미한다.

pwd

pwd는 print working directory의 약자로, 현재 위치 하고있는 경로를 출력하는 명령어 이다. 리눅스에서의 경로는 디렉터리를 의미하므로, pwd를 입력하면 해당 명령어를 입력한 디렉토리를 출력한다.

* 디렉토리란?

디스크에 기록되어 있는 파일 관리의 등록부를 말한다. 파일의 이름, 확장자, 크기, 작성일시, 작성시간 등이 기록되어 있는데, 파일의 읽기/쓰기(read/write)는 여기를 통하여 이루어지고 있다. 또, 계층 구조를 형성하고 있어 그 계층 자신을 가리켜 디렉터리라고도 한다. 오리지널이 되는 디렉터리(루트 디렉터리)의 밑에 수많은 디렉터리(서브 디렉터리)를 둘 수가 있고, 그룹별로 파일을 나누어 보존할 수 있도록 되어 있다. 디렉터리와 디렉터리 사이를 자유롭게 이동하여 이용할 수 있으며, 현재 위치 하고있는 디렉터리를 현행(current) 디렉터리라고 한다.



[리눅스 디렉터리 구조]

* 디렉토리 종류와 특징

- /(최상위 디렉토리, 루트)
- / root (최상위 디렉토fl 안에 root 디랙토리)
- 관리자의 홈 디렉토리는 /root 디렉토리
- 그 외 사용자의 홈 디렉토리는 /home 경로에 생성됨.

/(루트) - 리눅스 파일 체제의 최상위 디렉토리

루트 디렉토리라 한다. 모든 디렉토리들의 시작점이고 가장중요한 디렉토리이다. 파티션 설정 시 반드시 존재해야 하고 절대 경로의 기준이 된다.
* 절대 경로 - 시작이 /(root디렉토리) 부터 시작
* 상대 경로 - 시작이 자신의 디렉토리 부터 시작

/bin(바이너리): 기본적인 명령어가 저장된 디렉토리, 이진 파일(실행파일)

리눅스에서 자주 사용하는 mv, cp, rm 등과 같은 명령어들이 이 디렉토리에 존재한다. 이 디렉토리에 저장되어 명령어를 root 사용자들뿐만 아니라 일반 사용자들도 사용할 수 있다. cent OS 7버전부터는 /usr/bin으로 심볼릭 링크되어 있다.

* 심볼링 링크 - 링크를 연결해 원본 파일에 대한 참조를 한다. 즉, 링크를 연결해서 원본 파일을 직접 사용하는 것처럼 동작하는 링크라 보면 된다. 쉽게 말해 윈도우의 바로가기라고 생각하면 된다.

/boot : 리눅스 부트로더(Boot Loader)가 존재하는 디렉토리

GRUB와 같은 부트로더에 관한 파일들이 존재한다. 리눅스 커널은/ 디렉토리나/ boot 디렉토리에 존재해야 하며 별도의 파티션을 권장하고 있다. 리눅스 커널이 저장되어 있는 디렉토리이며, 각종 리눅스 부트에 필요한 부팅 지원 파일들이 저장되어있는 디렉토리 이다.

/dev(디바이스) : 시스템 디바이스(device)파일을 저장하고 있는 디렉토리

시스템의 각종 장치들이 접근하기 위한 장치 드라이버들이 저장되어 있는 디렉토리이다. 디렉토리를 살펴보면 많은 파일들이 위치하지만, 물리적인 용량을 갖지 않는 가상 디렉토리이다. 대표적으로 하드 드라이브, 플로피, CD-ROM, 루프백 장치 등이 존재한다.
리눅스 시스템은 윈도우와 달리 각종 장치들을 하나의 파일로 취급한다. 따라서 시스템은 각각의 장치로부터 정보를 /dev 디렉토리에 존재하는 해당 장치 파일로부터 가져온다. /dev/sda(하드디스크 장치 파일), /dev/cdrom(CD-ROM)장치파일 등과 같은 파일들이 여기에 위치한다.

/etc : 시스템의 거의 모든 환경 설정 파일이 존재하는 디렉토리

리눅스에 없어선 안되는 디렉토리 이며, 리눅스 시스템에 관한 각종 환경 설정에 연관된 파일들과 디렉토리를 가진 디렉토리이다. 바이너리 파일은 존재하지 않도록 한다. 이 디렉토리에 있는 대부분의 파일들은 시스템 관리자에 의해서 관리되는 파일들이다. 웹서버 환경설정, 시스템 초기화 설정 파일, TCP/IP 설정 파일 등 시스템 전반에 걸친 거의 모든 환경설정 파일들이 모두 이 디렉토리 안에 저장되어 있다.

/home : 사용자의 홈 디렉토리들이 위치하는 디렉토리

unseradd로 사용자를 생성하면 대부분/home 디렉토리 아래에 사용자 아이디와 동일한 이름의 디렉토리가 생성된다. 그리고 해당 디렉토리에 대한 접근 권한은 기본적으로 생성된 사용자만 접근 가능하다.

/mnt(마운트): 탈부착이 가능한 장치들에 대한 마운트 포인트로 사용하는 디렉토리

임시로 사용되는 디렉토리 이며, 일반적으로 모든 사용자들에게 열려있다.
직접 커맨드를 통해 마운트 할 때 사용되는 디렉토리이다,(마운트를 수동으로 해야만 한다.)

/lib(라이브러리),/lib64 :커널 모듈 파일과 프로그램들이 의존하고 있는 라이브러리 파일들이 존재하는 디렉토리

대부분 공유 라이브러리로 더 편하게 사용할 수 있으며, 파일의 크기를 줄여서 실행할 때 불러 사용하게 된다. CentOS 7버전부터는 /usr/lib에 심볼릭 링크 되어 있다. 64bit의 경우 /lib64 디렉토리가 있고 이 또한 CentOS 7버전 부터는 /usr/lib64에 심볼릭 링크 되어 있다.
+ 모듈 : 요청이 있으면 커널에 로드 혹은 언로드를 할 수 있는 코드 조각이다. 쉽게 말해 모듈을 커널이 시스템에 연결된 하드웨어에 엑세스 할 수 있도록 해주는 것이다. 라고 보면 된다.

/proc(프로세스) : 커널과 프로세스 정보를 위한 가상 파일 시스템

현재 메모리에 존재하는 모든 작업(각종 프로세스, 프로그램 정보, 하드웨어적인 정보)들이 파일 형태로 존재한다. 이 디렉토리는 디스크상에 실제 존재하는 것이 아니라 커널에 의해 메모리상에 존재하므로 가상 파일 시스템이라고 한다.
/dev 디렉토리와 마찬가지로 하드 디스크상에 물리적인 용량을 갖지 않는다. 실제로 존재하지 않는다는 것이다. 또한 이 디렉토리에 존재하는 파일들은 실제 하드 디스크에 저장되지 않고 커널에 의해서 메모리에 저장된다. 그러므로 그 디렉토리 안의 파일들은 현재의 시스템 설정을 보여주는 것이다.

/root: root 계정의 홈 디렉토리

/sbin(시스템 바이너리): 시스템 관리(부팅, 복원, 복구 및 보수)를 위한 명령어가 저장된 디렉토리

시스템 이진 파일(실행파일), ifconfig, ethtool, halt, e2fsck와 같은 시스템 명령어들을 저장하고 있는 디렉토리 시스템 관리를 위해서 사용되는 유틸리티들(사용자의 편의를 제공하는 실용적인 소프트웨어)과 기타 root만을 위한 명령어들이 /sbin, /usr/sbin, /usr/local/sbin에 저장된다. 이 중에서 /sbin 디렉토리는 /bin 디렉토리의 바이너리들에 더해 시스템의 부팅, 복원, 복구 및 수리하기 위한 중요한 바이너리들을 포함한다. CentOS 7 버전부터는 /usr/sbin 에 심볼릭 링크 되어 있다.
* 바이너리 : 0과, 1이 두 숫자로만 이루어진 이진법, 즉 시스템이 인식하고 실행될 수 있는 내용

/tmp: 임시 파일들이 저장되는 공간이고 공용 디렉토리

시스템 일반적인 사용자 또는 각종 프로세스에서 사용되는 파일들이 생성되는 위치이다. 재부팅 시 삭제되며 보통 10일마다 삭제된다.

/usr(유저): 주로 새로 설치되는 프로그램들이 저장되는 디렉토리
시스템 관리자와 프로그램 설체 및 관리와 관련되 있으며, 가장 중요한 디렉토리 중 하나로 사용자면서 공유할 수 있는 디렉토리와 파일들을 가지고 있다. 즉, 시스템에 사용되는 각종 응용 프로그램들이 설치되는 디렉토리이다. 일반 사용자들을 위한 대부분의 프로그램 라이브러리 파일들이 있다. /urs안에 존재하는 파일들은 모든 사용자들이 참조해서 사용할 수 있다. '윈도우의 Program Files 같은 폴더'

종류로는
* /usr/include: C 또는 C++ 프로그램에 의해 포함되는 헤더 파일들, 즉 시스템이 일반적으로 사용하는 include 파일들이 존재한다.
* /usr/lib: 프로그래밍과 패키지들을 위한 라이브러리 파일들이 존재하는 디렉토리
* /sr/local: 로컬 디렉토리 체계 해당 컴퓨터에서 사용할 소프트웨어를 시스템 관리자가 설치하는 디렉토리이다. /usr의 소프트웨어를 대체하거나 업그레이드 하기 위해 설치되는 소프트웨어는 이 디렉토리에 설치하도록 한다.
* /urs/share: 모든 읽기만 가능한 아키텍처 비 의존 자료들이 존재한다.
* /urs/src: 리눅스 커널 등의 소스코드가 포함된 디렉토리
* /usr/bin: 대부분의 사용자 명령어기 포함되어있다.
* /usr/sbin: 시스템 관리자에 의해서 사용되는 비중요 시스템 바이너리들이 존재한다. 시스템 수리, 시스템 복구, /usr 마운팅을 위해 필요한 시스템 관리 프로그램들 또는 다른 주요한 기능들이 /sbin을 대신해 존재한다.

[참고]
UnixOS에서 용량 문제로 꼭 필요한 파일과 그렇지 않은 파일을 분리해 관리하기 위해 bin과 sbin 디렉토리를 분산시켜 만들었다. 리눅스도 유니스 기반 OS기 때문에 bin과 sbin에 관련된 연관 디렉토리들이 있다. 연관 디렉토리 종류는
/usr/bin, /usr/sbin,/usr/local/bin, /usr/local/sbin 이 있다.

[기본적인 차이점]
* /bin: cd, ls 등의 사용자 커맨드 파일이 위치한 디렉토리 (필수적인 파일만 관리)
* /sbin: systemcl 들의 시스템 커맨드 파일이 우치한 디렉토리 (필수적인 파일만 관리)
* /usr/bin: 필요에 의해 설치된 사용자 커맨드 파일이 위치한 디렉토리(yum 등 패키지 관리)
* /usr/sbin: 필요에 의해 설치된 사용자 커맨드 파일이 위치한 디렉토리(yum등 패키지 관리자가 관리)
* /usr/local/bin: 기타 사용자 커맨드 파일이 위치한 디렉토리(사용자 또는 설치 파일이 해당 디렉토리에 파일 설치)
* /usr/local/sbin: 기타 시스템 커맨드 파일이 위치한 디렉토리(사용자 또는 설치파일이 해당 디렉토리에 파일 설치)

/val: 시스템에서 사용되는 동적 파일들이 저장되는 디렉토리
시스템 운용 중 생성되는 자료, 즉 변화되는 자료 파일들을 저장하기 위한 디렉토리 주로 로그들이 저장된다. (시스템 로그파일, 전송된 메일 임시 저장, 사용자 로그인의 보안 정보 등)

/opt: 추가적인 소프트웨어를 설치하는 디렉토리
대부분 추가적인 소프트웨어들이 /urs 디렉토리 아래에 설치된다. 응용프로그램 패키지 설치 저장소, 패키지 매니저가 자체적으로 설치/삭제를 수행한다.

/media: DVD, CD-ROM, USB등의 탈부착이 가능한 외부 장치들의 마운트 포인트로 사용하는 디렉토리
media 는 OS에서 관리해 주는 디렉토리.

/srv: 시스템에 의해 제공되는 데이터 파일 위치를 포함한 디렉토리
서비스들에 대한 자료 즉 FTP나 www 같은 특정

mkdir

mkdir은 make directory의 약자로, 현재 위치에서 새로운 디렉토리를 만들 때 사용한다. mkdir 만들어질_디렉토리의_이름의 형태로 사용하며, 여러 개의 디렉토리를 만들 때에는 만들어질 디렉토리의 이름을 공백으로 구분하여 나열한다.

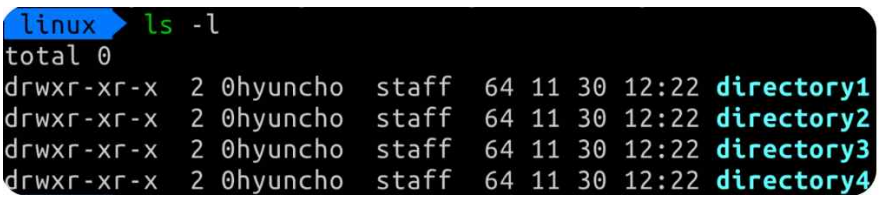
rmdir

rmdir은 remove directory의 약자로, 특정 디렉토리를 삭제할 때 사용한다. 마찬가지로, 여러 개의 디렉토리를 삭제할 때에는 공백으로 디렉토리를 구분하여 명령어 다음에 나열된다. 주의할 점은 rmdir로는 내용물이 비어있는 디렉토리만 삭제할 수 있다는 점이다. 내용물이 존재하는 디렉토리를 삭제할 때에는 이후에 소개할 rm 명령어를 사용한다.

ls

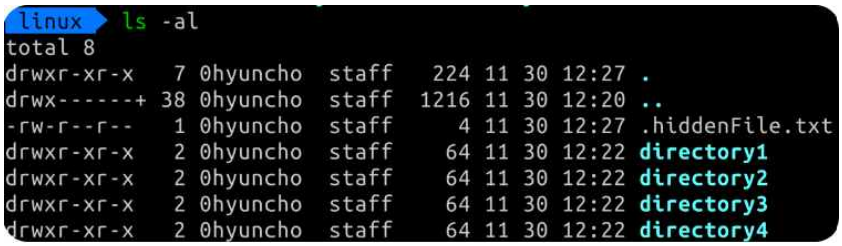
ls는 list의 약자로, 현재 위치한 디렉토리 내에 존재하는 모든 파일 및 하위 디렉토리의 목록을 출력해준다.

ls에는 다양한 옵션을 붙여 사용할 수 있는데, 대표적으로 -l과 -a가 많이 사용된다. ls -l을 입력하면 각 파일 및 디렉토리의 세부 정보까지 출력할 수 있다. 이 때 출력되는 세부 정보는 권한, 소유자, 그룹, 용량, 생성 시각 등의 정보를 포함한다.



또한, ls -a를 사용하면 숨김처리된 디렉토리들까지 목록 상에 조회할 수 있다. 참고로, 리눅스에서 파일명 앞에 .을 붙이면 해당 파일을 숨길 수 있다. 아래의 .hiddenFile은 숨겨진 파일 예시를 위해 생성한 텍스트 파일이다.

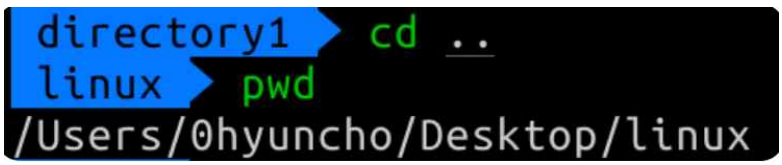
여러 개의 옵션을 조합해서 사용할 수 있다. 옵션을 조합할 때에는 - 옆에 사용하고자 하는 옵션을 의미하는 알파벳을 나열해준다. 참고로, ls -a와 ls -al의 결과를 보면 .과 ..을 확인할 수 있는데, .은 현재 위치한 디렉토리, ..은 현재 위치한 디렉토리의 상위 디렉토리를 의미한다.



cd

cd는 change directory의 약자로, 다른 디렉토리로 이동할 때 사용하는 명령어이다. 앞서 생성한 directory1로 이동하려면 아래와 같이 입력한다.

다시 이전의 디렉토리로 돌아가려면 cd ..을 입력하면 된다.



경로를 이동함에 있어 절대 경로와 상대 경로라는 개념이 있다. 절대 경로는 리눅스의 최상위 디렉토리인 루트 디렉토리(/)를 기준으로 특정 디렉토리의 경로를 지칭하는 개념으로, pwd를 입력했을 때 나타나는 경로가 절대 경로이다. 반면, 상대 경로는 현재 위치를 기준으로 타겟 디렉토리의 경로를 지칭한다.

예를 들어, directory1에 위치한 상태에서 directory2로 이동하고자 할 때, 아래와 같이 입력하면 입력한 경로는 절대 경로가 된다.

```
cd /Users/0hyunCho/Desktop/linux/directory2
```

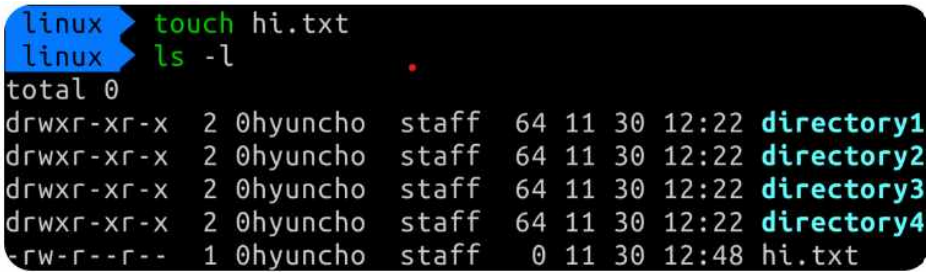
반면, 아래와 같이 현재 위치를 기준으로 경로를 입력하면 상대 경로가 된다.

```
cd ../directory2
```

touch

touch는 본래 파일의 생성 날짜 및 시각을 수정할 때에 사용하는 명령어지만, 내용물이 비어 있는 파일을 생성할 때에도 사용한다. 내용물이 존재하는 파일을 생성할 때에는 vim이나 nano 등의 텍스트 편집기를 일반적으로 사용하지만, 텍스트 편집기의 종류가 다양하다.

아래와 같이 입력하면, txt 확장자를 가진 hi라는 이름의 텍스트 파일이 생성된다. 이 텍스트 파일의 용량은 0으로, 내용물이 비어있다.



여기에서 pwd의 실행 결과를 hi.txt에 저장해보자. 어떤 명령어의 실행 결과를 파일 내에 저장할 때에는 >를 사용한다. 아래와 같이 입력하면 아무 것도 뜨지 않는데, 리눅스에서 명령어를 입력했을 때 아무 것도 출력되지 않으면 이는 해당 명령어의 수행이 성공했음을 의미한다.

Cat

cat은 concatenate의 약자로, 여러 파일들의 내용을 연결하여 출력시켜준다. 아래와 같이 입력하면 앞서 만든 hi.txt를 출력할 수 있다. 특정 파일의 내용을 간단하게 확인해보고자 할 때에 많이 사용된다.

mv

mv는 move의 약자로, 파일을 이동시킬 때와 파일의 이름을 변경할 때 사용한다. 먼저 파일을 이동시킬 때에는 mv 이동_대상_파일 이동시키고자_하는_디렉토리의 형태로 아래와 같이 명령어를 입력해주면 된다.

파일의 이름을 변경하고자 할 때에는 mv 변경_대상_파일_이름 변경될_파일_이름의 형태로 명령어를 입력해준다.

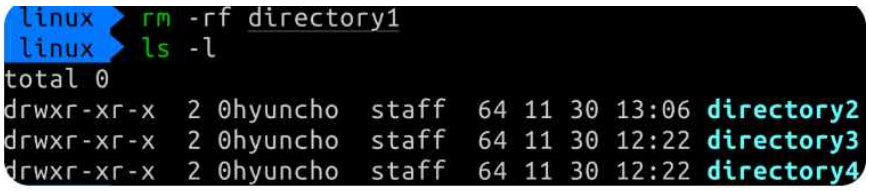
cp

cp는 copy의 약자로, 파일을 다른 위치에 복사하고자 할 때에 사용합니다. 현재 directory1에 위치해 있는 hello.txt를 directory2로 이동시키려면 아래와 같이 명령어를 입력하면 된다. cp는 파일을 복제한 새로운 파일을 생성하는 것이므로, 원본 파일을 삭제하지 않는다. 즉, directory1과 directory2에 동일한 내용을 가진 파일이 존재하게 된다.

rm

rm은 remove의 약자로, 파일을 삭제할 때 사용하는 명령어이다. 앞서 복사한 hello.txt를 삭제하려면 아래와 같이 명령어를 입력하면 된다.

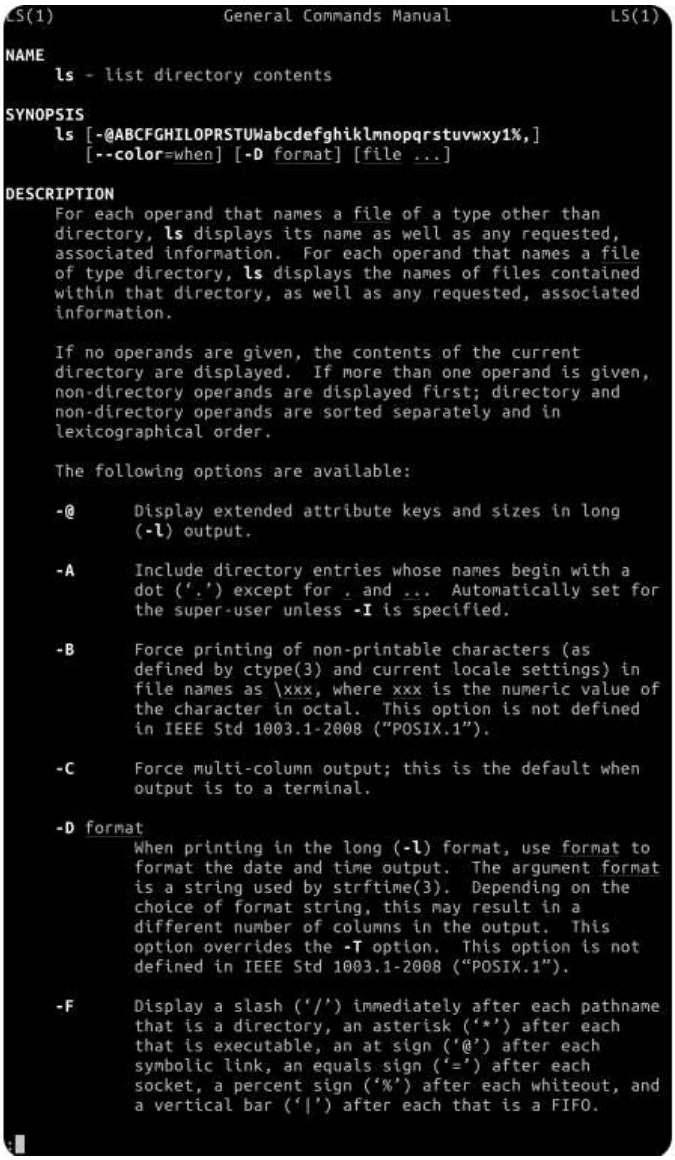
rm 명령어에는 -rf 옵션을 붙일 수 있다. 여기에서 -r 옵션은 recursive의 약자로, rm의 동작을 재귀적으로 수행하라는 것을 의미하며, -f은 forced의 약자로 삭제 확인 과정을 거치지 않을 때 사용한다. 즉, rm -rf directory1과 같이 입력하면 directory1을 포함하여, 그 안에 있는 모든 파일 및 하위 디렉토리를 묻지도 따지지도 말고 삭제하라는 의미가 된다. 따라서, 내용물이 존재하는 디렉토리를 삭제할 때에 rm -rf를 사용한다.



rm -rf를 사용하면 휴지통을 거치지 않고 시스템에서 바로 삭제가 진행되니 해당 명령어를 사용할 때에는 주의가 필요하다.

man

man은 manual의 약자로, 리눅스에서 사용할 수 있는 모든 명령어의 사용법을 출력시킬 때 사용한다. 예를 들어, ls 명령어에 붙일 수 있는 옵션을 알아보고 싶거나, cp 명령어의 자세한 사용법을 알고 싶은 경우에 man 명령어를 사용하면 된다. man 명령어_이름의 형태로 사용하며, 명령어를 입력하면 아래와 같이 매뉴얼이 열린다.



ls 명령어와 관련된 설명과 사용할 수 있는 모든 옵션이 매뉴얼에 정리돼 있는 것을 확인할 수 있다. 위아래로 스크롤 하거나, 위아래 방향키를 활용하여 매뉴얼 문서 내에서 이동할 수 있으며, 필요한 내용을 확인한 후에는 q를 눌러 매뉴얼 창을 닫을 수 있다.

Git

- 1. git이 중요한 이유

*** git의 의미**

-> ["_*깃([길])*은 컴퓨터 파일의 변화를 추적하고 여러 사람들 사이에서 그 파일을 조정하는 버전 관리 시스템입니다._"]

이것은 다시 말해서 깃의 가장 기본적이고 중요한 기능이 여러 팀에게 동시에 같은 프로젝트로 코드를 더하게 (그리고 통합)하게 허용한다는 걸 의미한다. 이러한 능력을 프로젝트에 더함으로써 팀을 더 효율적이게 하고 더 큰 규모의 프로젝트나 더 복잡한 문제들에 일할 수 있는 능력을 제공한다.

깃이 정말 잘하는 또 다른 많은 것들이 있습니다: 변경점을 원복, 새로운 기능을 더하기 위한 새 브랜치(branches) 생성, 병합 시 발생하는 충돌(conflict) 해결 등을 허용한다.

- 2. git이 동작하는 법

깃은 저장소(repositories) 안에 프로젝트를 저장한다. **커밋(Commits)**이 프로젝트로 만들어지면 깃에게 여러분이 생성한 새로운 코드나 변경한 코드에 대해 만족하는지 말해줄 것이다.

새로운 코드/변경점들은 브랜치에 커밋 된다. 대부분의 일은 다른 브랜치에 커밋되고 그 다음 마스터 브랜치로 병합된다. 이에 대한 모든 것은 프로젝트로 같은 디렉토리(directory) 안이지만 .git이라고 불리는 하위 폴더 안에 저장된다.

동료와 코드를 공유하기 위해서는 여러분이 저장소로 변경점을 **푸쉬(push)**한다. 동료로부터 새로운 코드를 얻어오려면 저장소로부터 변경점을 **풀(pull)**한다.

- 3. 명령어

새로운 깃 저장소 초기 설정: Git init

모든 코드 내용은 저장소 안에 추적된다. 깃 저장소를 초기 설정하려면 해당 프로젝트 폴더 안에 이 명령어를 사용하면. git 폴더를 만들어 줄 것이다.

```
git init
```

Git add

이 명령어는 하나 또는 모든 변경 파일들을 본 무대 영역으로 더합니다.
어떤 하나의 특정 파일을 올리기 위해서는:

```
git add filename.py
```

신규 또는 수정되거나 삭제된 파일들을 올리기 위해서는:

```
git add -A
```

신규 또는 수정 파일들을 올리기 위해서는:

```
git add .
```

수정 또는 삭제된 파일들을 올리기 위해서는:

```
git add -u
```

Git commit

이 명령어는 버전 이력을 파일 안에 기록한다. -m이 뜻하는 것은 어떤 커밋 메세지가 뒤따른다는 의미한다. 이 메세지는 커스텀이며 반드시 이것을 동료에게 알리는 용도로 또는 미래의 스스로에게 무엇이 해당 커밋 안에 더해졌는지 알리기 위해 사용해야 한다.

```
git commit -m "your text"
```

Git status

이 명령어는 파일들을 초록색과 빨간색으로 리스트해 줄 것이다. 초록색 파일들은 무대로 올려졌지만 아직 커밋되지 않은 것들이다. 빨간색으로 표시된 파일들은 무대로 아직 올려지지 않은 것들이다.

```
git status
```

브랜치에 작업하는 것

Git branch branch_name

이것은 새 브랜치를 생성합니다:

```
git branch branch_name
```

* 브랜치란?

모든 버전 관리 시스템은 브랜치를 지원한다. 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.

사람들은 브랜치 모델이 Git의 최고의 장점이라고, Git이 다른 것들과 구분되는 특징이라고 말한다. 당최 어떤 점이 그렇게 특별한 것일까. Git의 브랜치는 매우 가볍다. 순식간에 브랜치를 새로 만들고 브랜치 사이를 이동할 수 있다. 다른 버전 관리 시스템과는 달리 Git은 브랜치를 만들어 작업하고 나중에 Merge 하는 방법을 권장한다. 심지어 하루에 수십 번씩해도 괜찮다. Git 브랜치에 능숙해지면 개발 방식이 완전히 바뀌고 다른 도구를 사용할 수 없게 된다.

Git checkout -b branch_name

새로운 브랜치를 생성하고 그것으로 자동 전환하려면:

```
git checkout -b branch_name
```

이것은 간략하게:

```
git branch branch_name
git checkout branch_name
```

Git branch

모든 브랜치를 리스트하고 현재 어떤 브랜치에 있는지 확인하려면:

```
git branch
```

Git log

이 명령어는 현재 브랜치에서 모든 버전 이력을 리스트해줄 것입니다:

```
git log
```

Push와 Pull

Git push

이 명령어는 커밋된 변경점들을 원격 저장소에 보냅니다:

```
git push
```

Git pull

원격 저장소에서 개인 컴퓨터로 변경점들을 가져오려면:

```
git pull
```

커밋되지 않은 모든 변경점을 보낼 때

문자 그대로 이 명령어는 커밋되지 않은 모든 변경점을 보냅니다:

```
git reset --hard
```

커밋 메세지 수정

커밋 메세지를 수정하는 건 굉장히 쉽습니다:

```
git commit --amend -m "New message"
```

개인 컴퓨터에서는 삭제하지 않고 깃에서만 파일을 삭제

종종 "git add" 명령어를 사용하면서 추가하지 않고 싶어했던 파일을 더해버릴 수도 있습니다.

"git add"를 사용하는 동안 주의하지 않는다면 커밋하기 원하지 않았던 파일을 더하게 될지도 모릅니다. 파일의 기록 버전(staged version)을 반드시 삭제해야 하고, 그리고나서 파일에 .gitignore를 추가하여 같은 실수를 두번 하는 일을 방지할 수 있습니다:

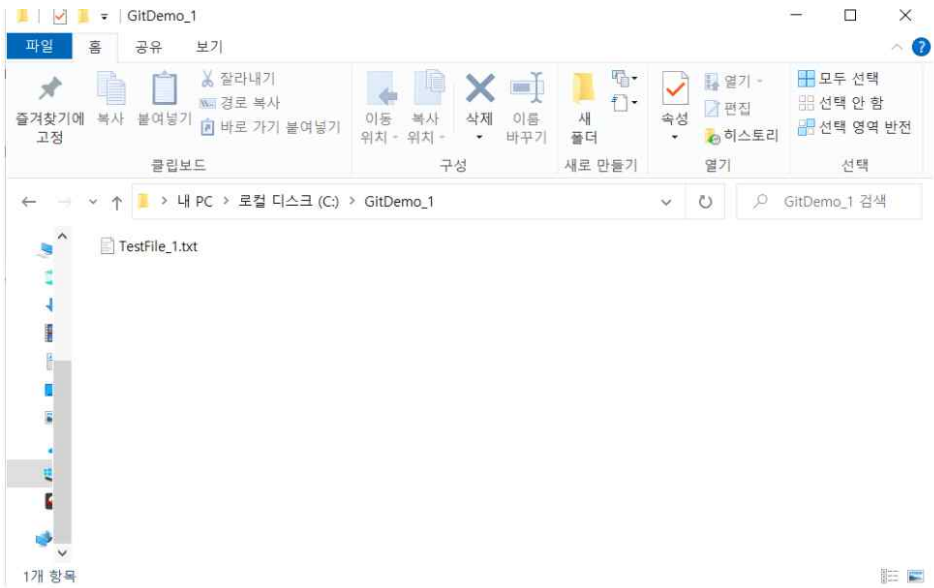
```
git commit --amend -m "New message"
```

- 4. Git repository 만들기

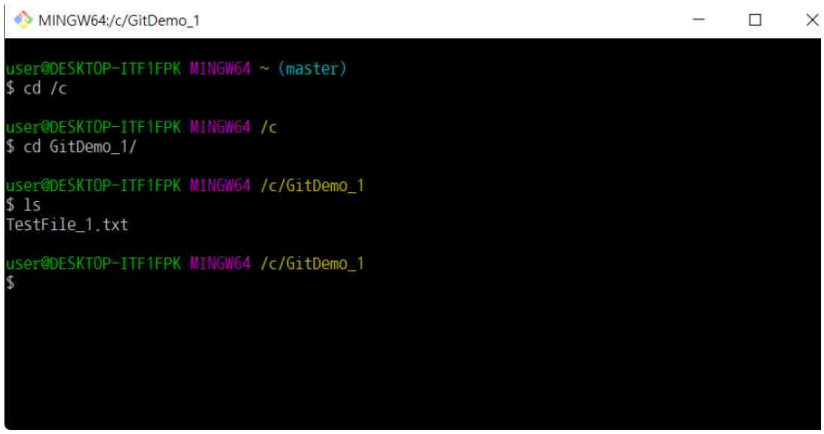
* 작업물을 레파지토리(저장소)에 올리는 방법 3가지 방법

1. 작업하던 폴더를 init> git 레파지토리 생성 > 연결(remote add)
2. git 레파지토리 생성 > clone > clone 받은 폴더안에 작업
3. 유니티 프로젝트 생성 > git 레포지토리 생성

1. 작업하던 폴더를 init> git 레파지토리 생성 > 연결(remote add)



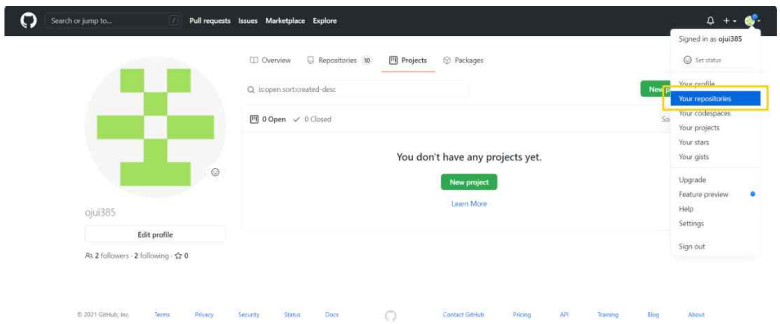
위 와 같은 폴더와 작업물이 있다고 가정



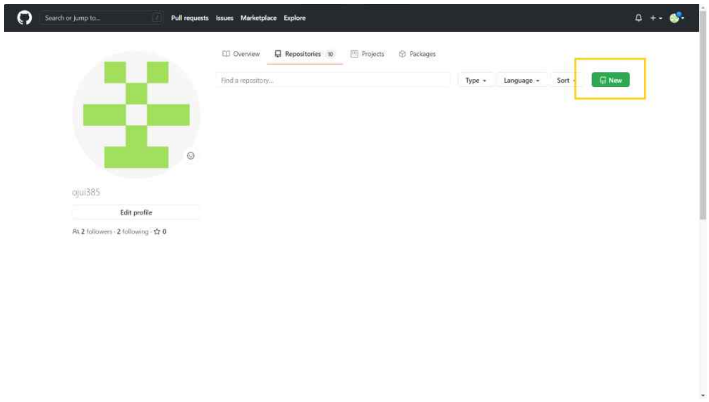
Git Bash를 열어 작업하던 폴더로 진입

- cd/c : c드라이브로 진입
- cd 폴더이름 : 현재 위치한 곳에서 폴더디름을 찾아서 진입
- ls : 현재 위치한 곳에 있는 파일 목록을 보여줌

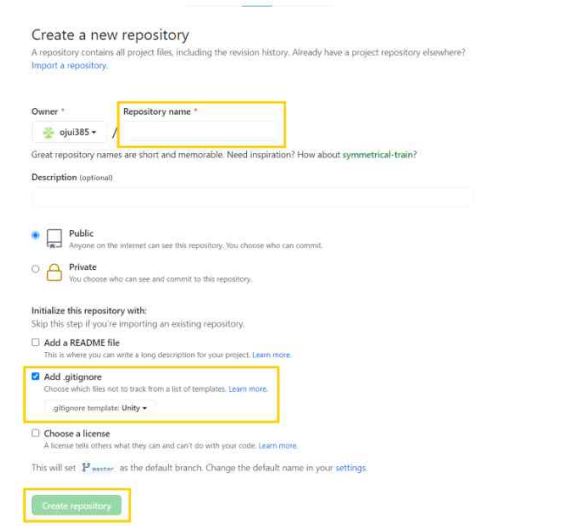
git init : 로컬저장소 생성



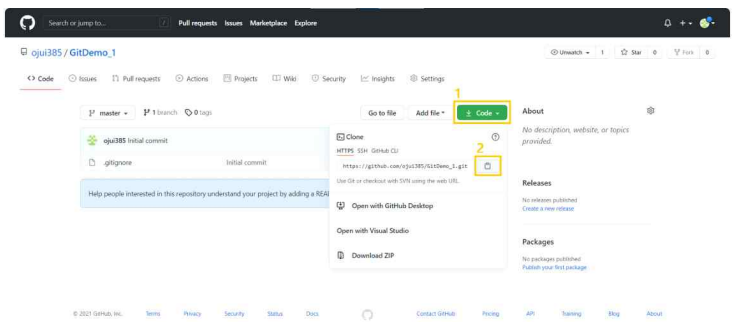
github 홈페이지 > 로그인
your repositories



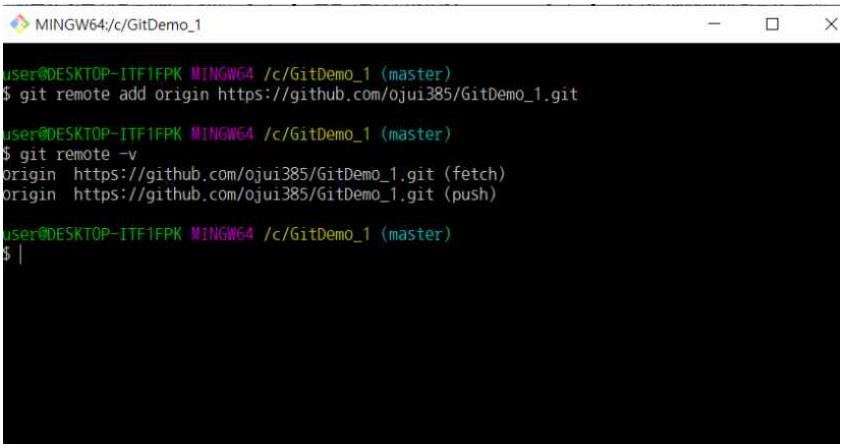
- New 클릭



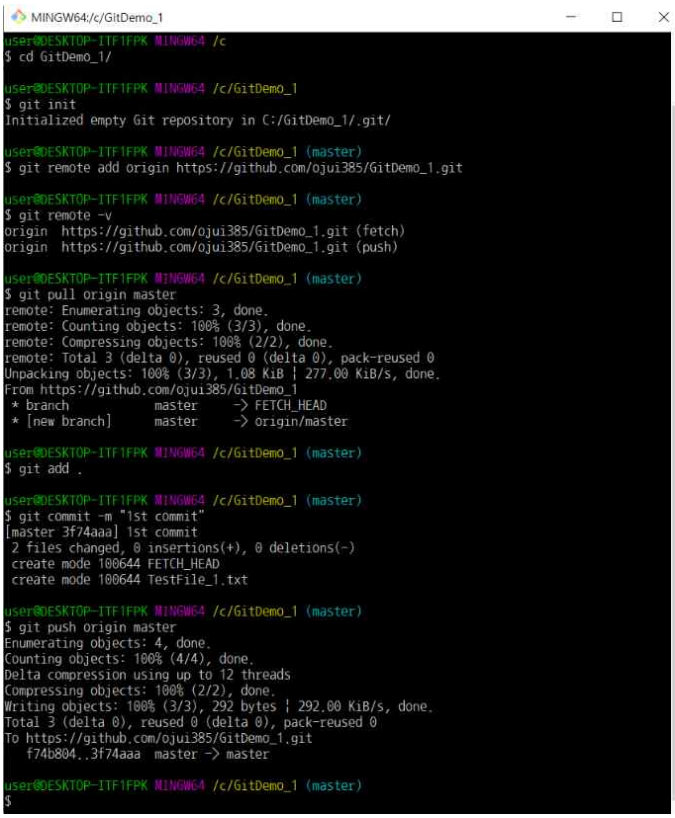
- 저장소 이름 기입
- public / private : 남들에게 보여줄지, 나만 볼지 설정
- ADD . gitgnore > Unity : unity 프로젝트를 올릴 거라면 설정



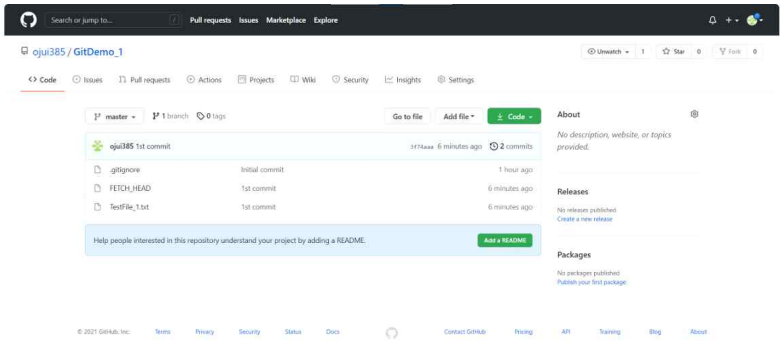
- code
- 주소 복사



- git remote add origin 카피한 주소 : 원격 저장소와 연결
- git remote -v : 연결된 원격 저장소 확인



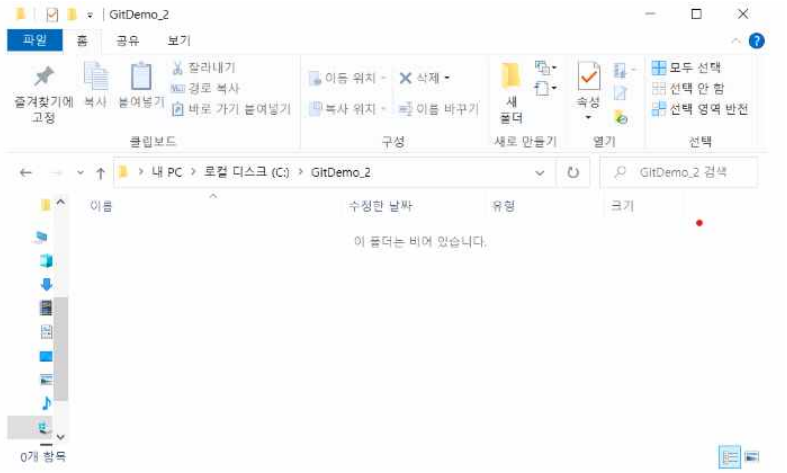
- git pull origin master : 원격저장소의 master 브랜치를 pull하여 로컬의 master 브랜치와 연결
- git add : 모든 파일을 스테이지에 올림
- git commit-m“커밋메시지” : 스테이지에 올린 파일들을 커밋
- git push origin master : 원격저장소의 master 브랜치에 push



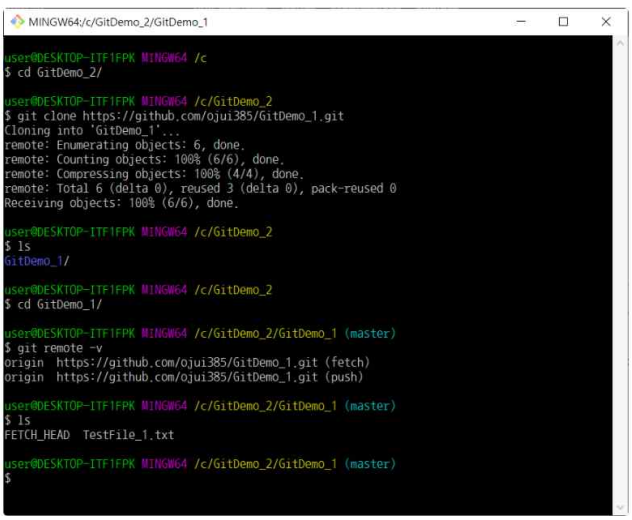
- 커밋이 올라온 것을 확인할 수 있음

2. git 레파지토리 생성 > clone > clone 받은 폴더 안에 작업

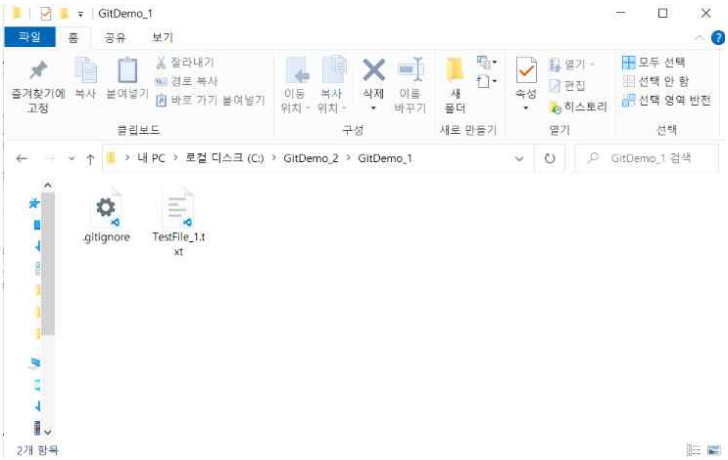
- 앞서 만들어 놓은 원격 저장소를 이용
- 주소 복사



- 원격 저장소를 clone할 위치



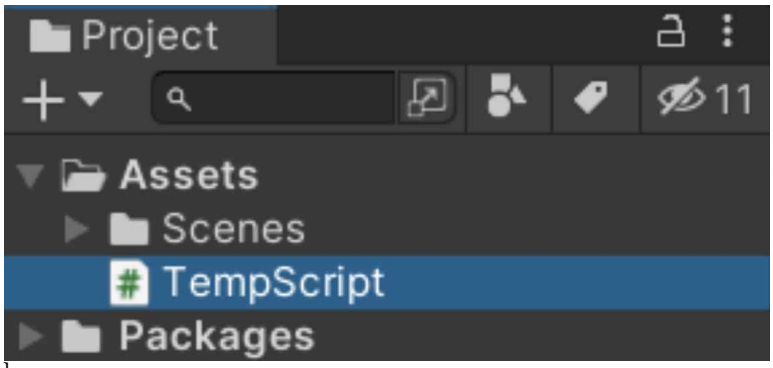
- git clone 카피한 주소



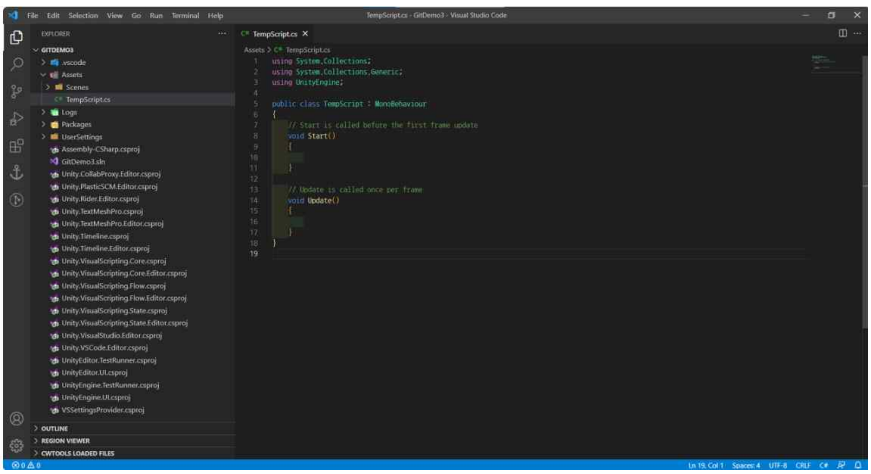
- 클론한 저장소의 이름으로 폴더가 생성 ehlTrh, 그아래에 저장소이 있던 파일들이 생성된 것을 확인

3. 유니티 프로젝트 생성 > git 레포지토리 생성

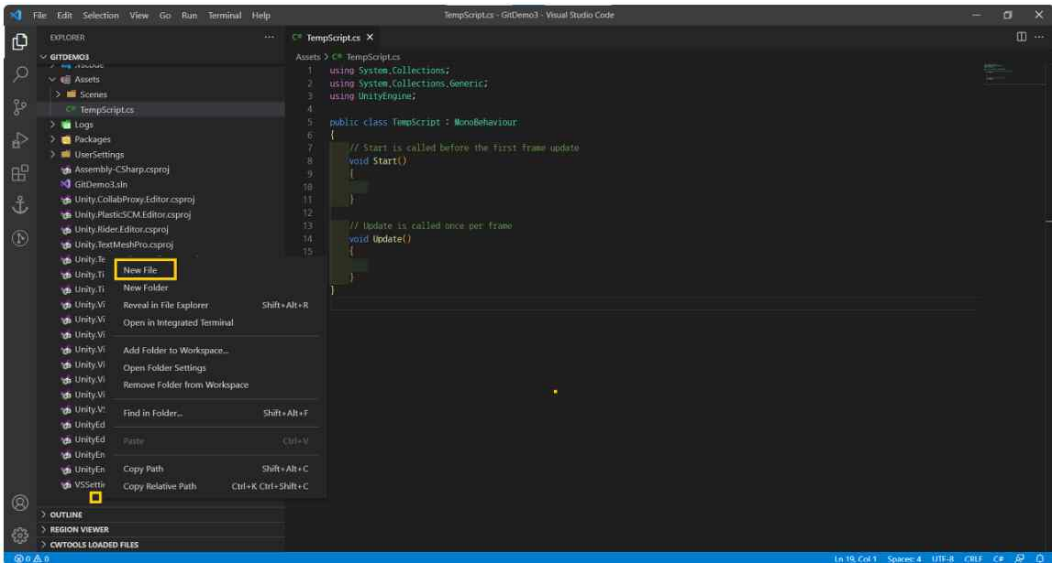
- 유니티 프로젝트 생성



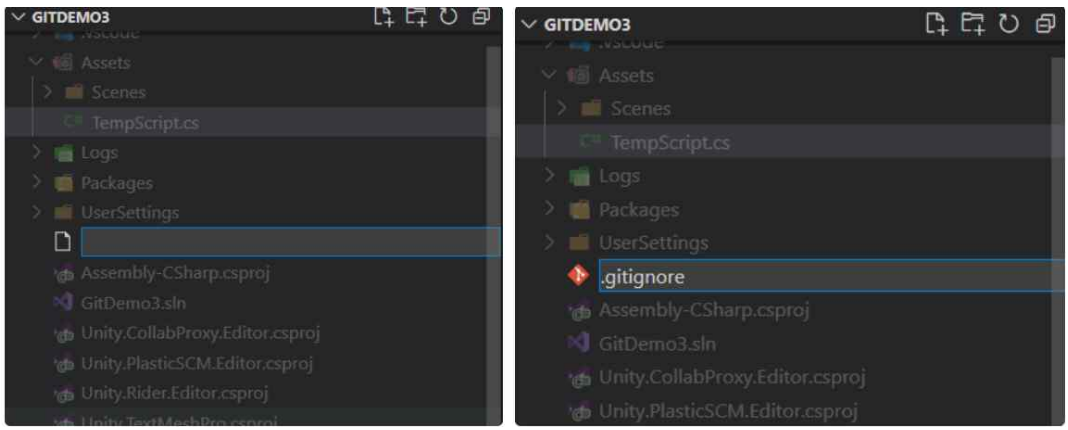
- script 생성



- vscode로 열기



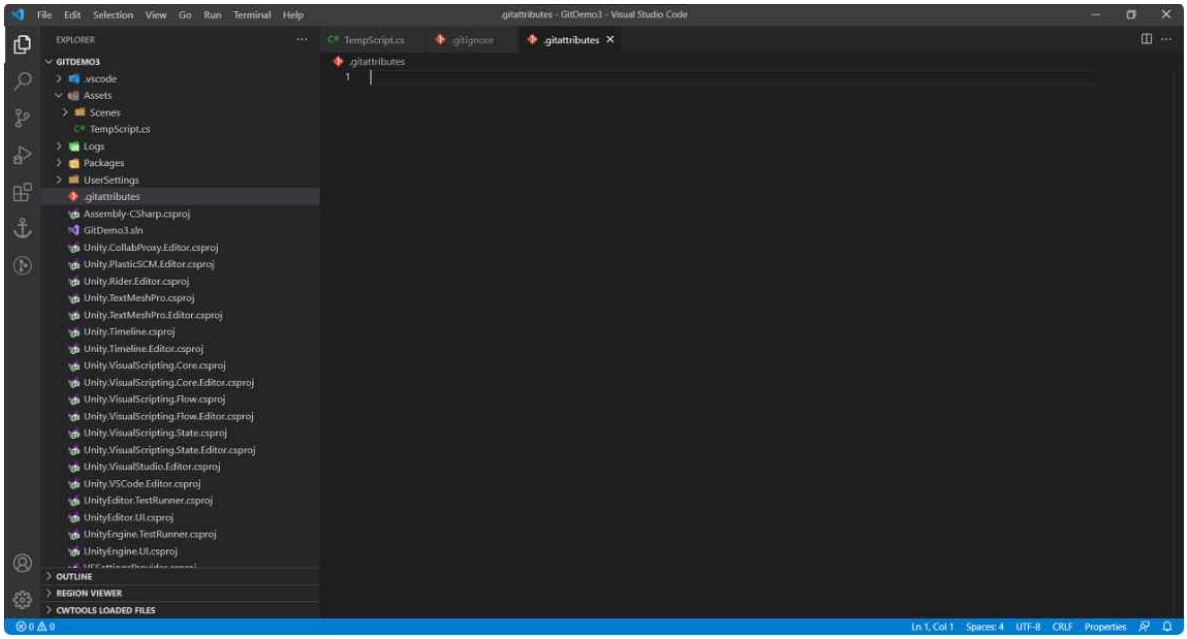
- 우클릭 > New File
- 주의 : 우클릭 시 폴더 안에 있는 구역이 아니라 맨 아래에 있는 빈공간에 우 클릭을 해야 함,



- 파일명 : .gitignore
- 제대로 입력하면 우측 아이콘이 변함

```
# Created by https://www.toptal.com/developers/gitignore/api/unity
# Edit at https://www.toptal.com/developers/gitignore?templates=unity

### Unity ###
# This .gitignore file should be placed at the root of your Unity project directory
#
# Get latest from https://github.com/github/gitignore/blob/master/Unity.gitignore
/[Ll]ibrary/
/[Tt]emp/
/[Oo]bj/
/[Bb]uild/
/[Bb]uilds/
/[Ll]ogs/
/[Uu]ser[Ss]ettings/

# MemoryCaptures can get excessive in size.
# They also could contain extremely sensitive data
/[Mm]emoryCaptures/

# Asset meta data should only be ignored when the corresponding asset is also ignored
!/[Aa]ssets/**/*.meta

# Uncomment this line if you wish to ignore the asset store tools plugin
# /[Aa]ssets/AssetStoreTools*

# Autogenerated Jetbrains Rider plugin
/[Aa]ssets/Plugins/Editor/JetBrains*

# Visual Studio cache directory
.vs/

# Gradle cache directory
.gradle/

# Autogenerated VS/MD/Consulo solution and project files
ExportedObj/
.consulo/
*.csproj
*.unityproj
*.sln
*.suo
*.tmp
*.user
*.userprefs
*.pidb
*.booproj
*.svd
*.pdb
*.mdb
*.opendb
*.VC.db

# Unity3D generated meta files
*.pidb.meta
*.pdb.meta
*.mdb.meta

# Unity3D generated file on crash reports
sysinfo.txt

# Builds
*.apk
*.aab
*.unitypackage

# Crashlytics generated file
crashlytics-build.properties

# Packed Addressables
/[Aa]ssets/[Aa]ddressable[Aa]ssets[Dd]ata/*/*.bin*

# Temporary auto-generated Android Assets
/[Aa]ssets/[Ss]treamingAssets/aa.meta
/[Aa]ssets/[Ss]treamingAssets/aa/*

# End of https://www.toptal.com/developers/gitignore/api/unity
```

- gitignore 파일에 위의 코드를 붙여넣기
- 저장



- 똑같이 .gitattributes 파일 생성

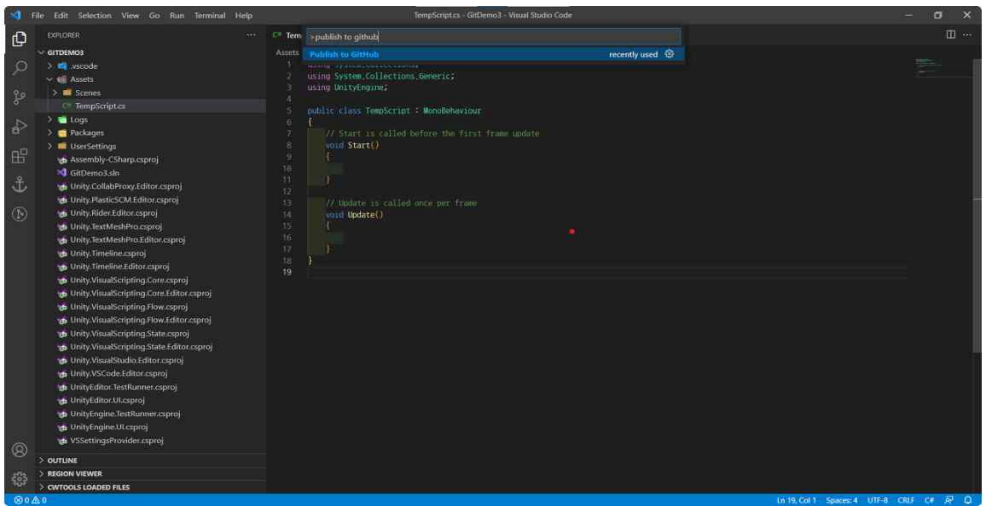
```
# Unity
*.cginc             text
*.cs                text diff=csharp
*.shader            text

# Unity YAML
*.mat               merge=unityyamlmerge eol=lf
*.anim              merge=unityyamlmerge eol=lf
*.unity             merge=unityyamlmerge eol=lf
*.prefab            merge=unityyamlmerge eol=lf
*.asset             merge=unityyamlmerge eol=lf
*.meta              merge=unityyamlmerge eol=lf
*.controller        merge=unityyamlmerge eol=lf

# "physic" for 3D but "physics" for 2D
*.physicMaterial2D  merge=unityyamlmerge eol=lf
*.physicMaterial    merge=unityyamlmerge eol=lf
*.physicsMaterial2D merge=unityyamlmerge eol=lf
*.physicsMaterial   merge=unityyamlmerge eol=lf

# Using Git LFS
# Add diff=lfs merge=lfs to the binary files

# Unity LFS
*.cubemap           binary
*.unitypackage      binary
```

```
# 3D models
*.3dm                binary
*.3ds                binary
*.blend              binary
*.c4d                binary
*.collada            binary
*.dae                binary
*.dxf                binary
*.FBX                binary
*.fbx                binary
*.jas                binary
*.lws                binary
*.lxo                binary
*.ma                 binary
*.max                binary
*.mb                 binary
*.obj                binary
*.ply                binary
*.skp                binary
*.stl                binary
*.ztl                binary

# Audio
*.aif                binary
*.aiff               binary
*.it                 binary
*.mod                binary
*.mp3                binary
*.ogg                binary
*.s3m                binary
*.wav                binary
*.xm                 binary
```

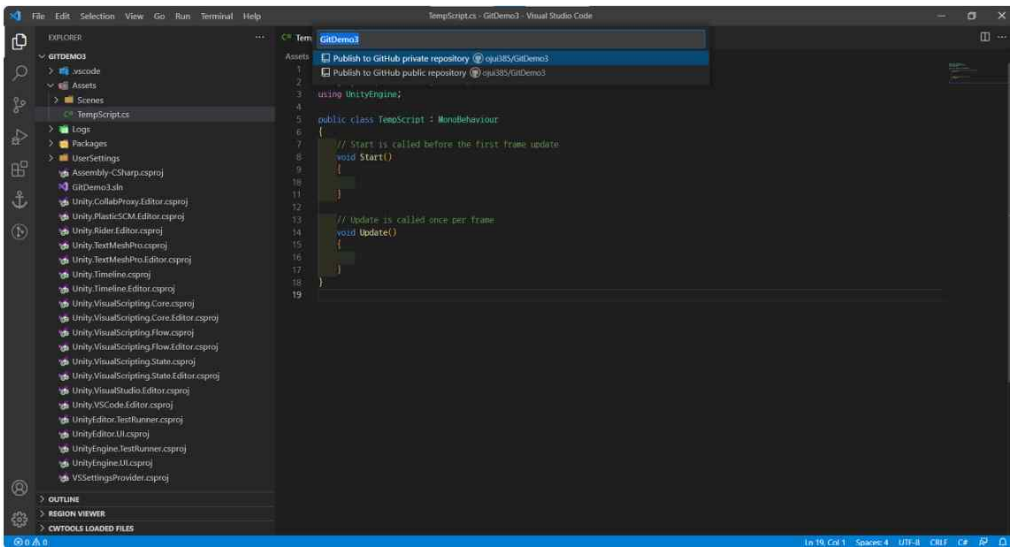
```
# Video
*.asf                binary
*.avi                binary
*.flv                binary
*.mov                binary
*.mp4                binary
*.mpeg               binary
*.mpg                binary
*.ogv                binary
*.wmv                binary

# Images
*.bmp                binary
*.exr                binary
*.gif                binary
*.hdr                binary
*.iff                binary
*.jpeg               binary
*.jpg                binary
*.pict               binary
*.png                binary
*.psd                binary
*.tga                binary
*.tif                binary
*.tiff               binary
```

```
# Compressed Archive
*.7z                 binary
*.bz2                binary
*.gz                 binary
*.rar                binary
*.tar                binary
*.zip                binary

# Compiled Dynamic Library
*.dll                binary
*.pdb                binary
*.so                 binary

# Fonts
*.otf                binary
*.ttf                binary

# Executable/Installer
*.apk                binary
*.exe                binary

# Documents
*.pdf                binary

# ETC
*.a                  binary
*.rns                binary
*.reason             binary

# Collapse Unity-generated files on GitHub
*.asset              linguist-generated
*.mat                linguist-generated
*.meta               linguist-generated
*.prefab             linguist-generated
*.unity              linguist-generated

# Spine export file for Unity
*.skel.bytes         binary
```

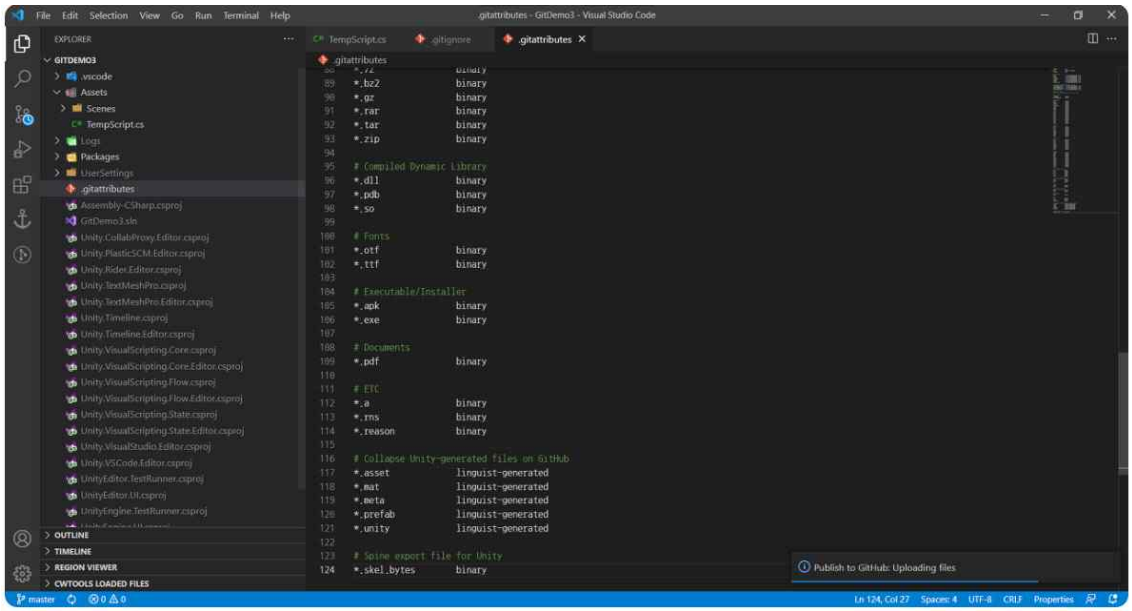
- gitattributes 파일에 위의 코드 붙여넣기
- 저장

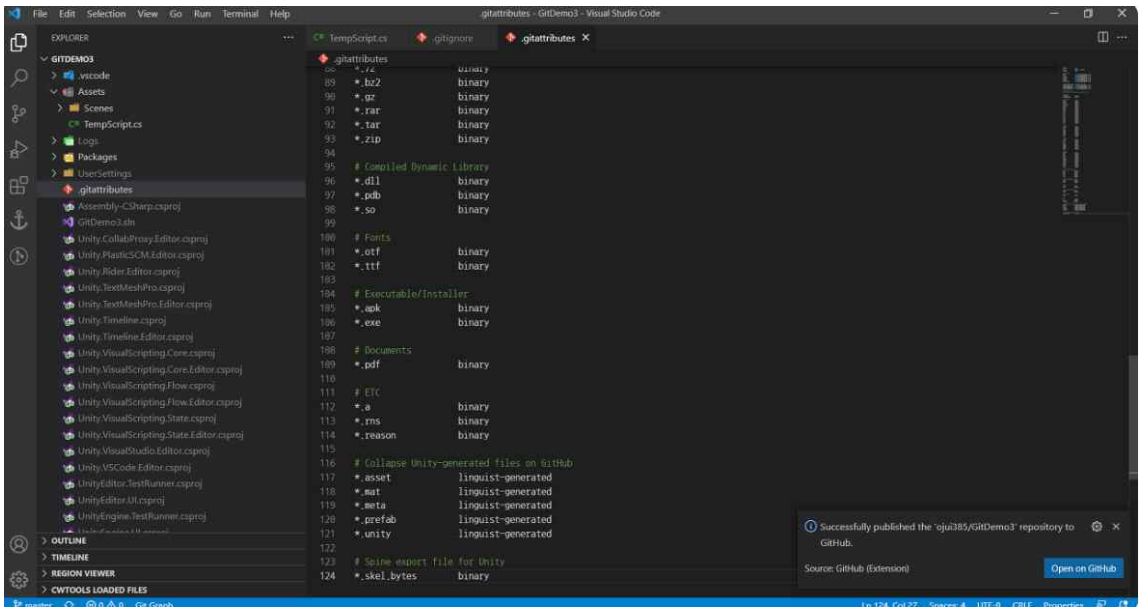


- Ctrl + Shift + P
- Publish to Gitub

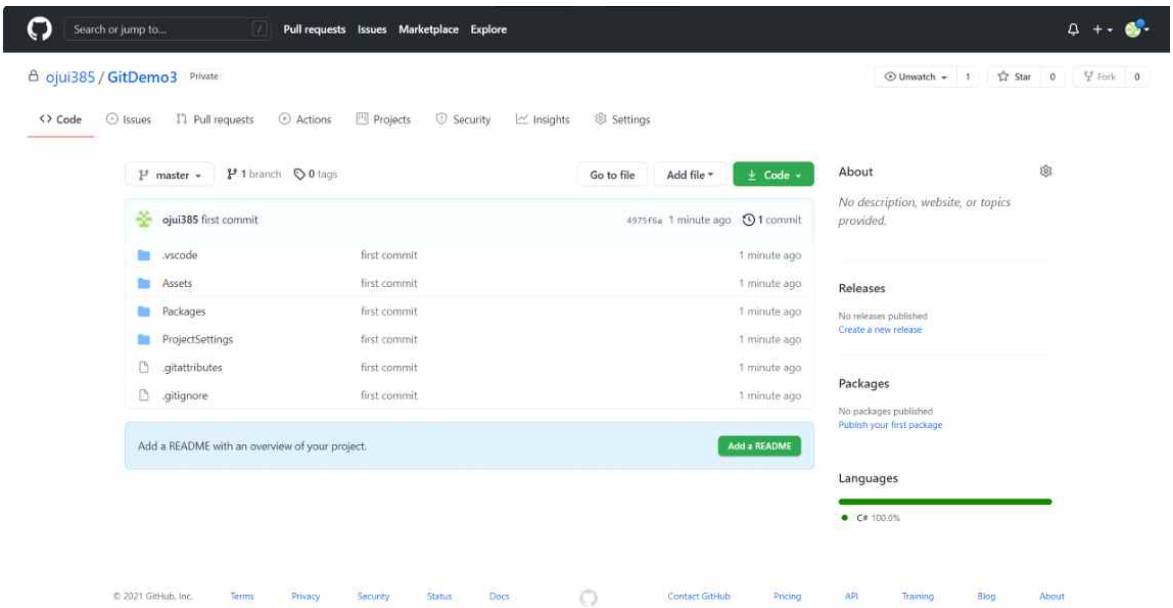


- private / public 결정





- 우측 하단에 뜨는 팝업창 : 로딩이 끝나면 원격저장소가 생성됐다는 문구가 뜸



- 유니티프로젝트가 포함된 레파지토리가 생성된 것을 확인