**<리눅스, GIT 보고서>**

로빛 18기 인턴 이선경

**<목차>**

**1. 리눅스**

**1) 운영체제 리눅스**

-개념  
-커널이란?

**2)리눅스(우분투) 터미널 명령어 정리**  
 -주로 활용되는 명령어

- 리눅스 특수문자 정리

-조금 더 편리해지는 터미널 창 도구들

**3) 명령어 정리**  
  
**2. GIT**

**1) 깃이란?**

-장점

-구조

-기본 용어  
 **2) Git 설치 및 사용**  
 - 초기 설정

- 파일 업로드

- 파일 불러오기

- 원격저장소 변경

**3) Git 명령어 정리**

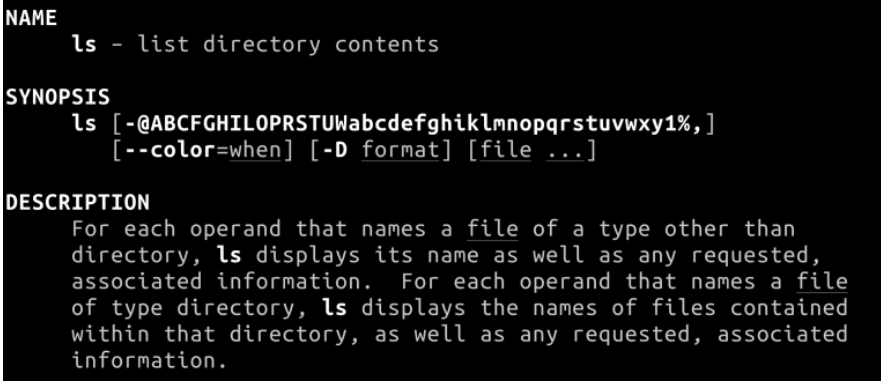
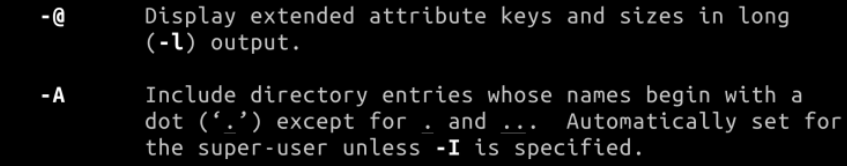
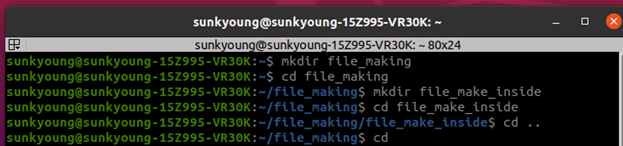
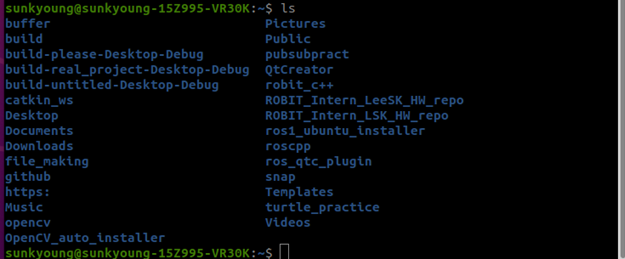
**[1. 리눅스]**

1. **운영체제 리눅스**   
    : 운영체제란, 오퍼레이팅 시스템(operating system)으로 사용자의 [하드웨어](https://ko.wikipedia.org/wiki/%ED%95%98%EB%93%9C%EC%9B%A8%EC%96%B4), [시스템 리소스](https://ko.wikipedia.org/wiki/%EC%8B%9C%EC%8A%A4%ED%85%9C_%EB%A6%AC%EC%86%8C%EC%8A%A4)를 제어하고 [프로그램](https://ko.wikipedia.org/wiki/%EC%BB%B4%ED%93%A8%ED%84%B0_%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%A8)에 대한 일반적 서비스를 지원하는 [시스템 소프트웨어](https://ko.wikipedia.org/wiki/%EC%8B%9C%EC%8A%A4%ED%85%9C_%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4)이다. 리눅스는 오픈소스 운영체제의 한 종류로, 무료로 공개되어 누구나 활용할 수 있다. 리눅스의 가장 대표적인 특징은 커널형태로 만들어진 오픈소스로 사용자가 원하는 대로 커스터마이징이 가능하다는 점이다. 이런 커스터마이징 가능한 리눅스의 여러 배포버전에는 대표적으로는 리눅스 기반으로 많은 사람들이 참여해 꾸준히 보완하며 만들어지고 있는 우분투가 있다.

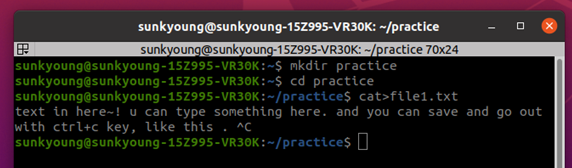
**커널이란**?  
리눅스 운영 체제의 주요 요소이자 컴퓨터 하드웨어와 프로세스를 잇는 인터페이스이다. 이 두 관리 리소스 사이에서 최대한 효과적으로 통신하게 된다. 이 커널은 운영체제 내부에 위치하여 서버나 컴퓨터 유형에 관계없이 하드웨어의 모든 주요 기능을 제어하게 된다.   
  
커널의 기능은 4가지가 있다.   
1) 메모리 관리  
2) 프로세스 관리  
3) 장치 드라이버   
4) 시스템 호출 및 보안  
메모리가 어디에서 무엇을 저장하는데에 얼마의 공간이 활용되는지를 추적하며 프로세스들이 어떤 프로세스가 CPU를 어느 시점에 얼마나 오래 활용할지를 결정하는 역할을 맡는다. 또한 하드웨어와 프로세스 사이의 중재, 인터프리터 역할을 수행하며 시스템 호출과 보안으로는 프로세스의 서비스 요청을 수신하게 된다.

**2. 리눅스(우분투) 터미널 명령어 정리**: 터미널 창은 Ctrl + Alt + T 로 열 수 있다.

**<주로 활용되는 명령어>**

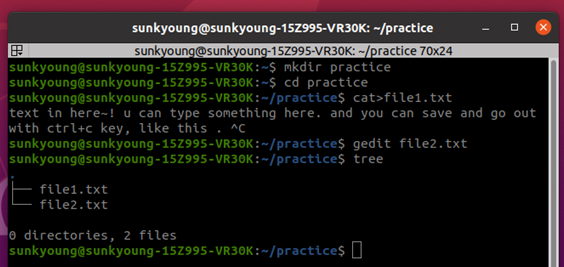
1. **Man**: manual의 약자로, 리눅스에서 사용할 수 있는 모든 명령어의 사용법을 출력시킬 때 사용합니다. 예를 들어, Is 명령어에 붙일 수 있는 옵션을 알아보고 싶거나, cp 명령어의 자세한 사용법을 알고 싶은 경우에 man 명령어를 사용하면 됩니다. man 명령어\_이름의 형태로 사용한다.   
   예를들어 $ man ls 으로 입력하면   
     
   위와같이 해당 이름과 사용방법, 개요, 그리고 함께올 수 있는 옵션들또한 설명해준다.
2. **Mkdir**: 현재 위치에서 새로운 디렉토리를 만들 때 활용한다.  
   - **mkdir 만들어질\_디렉토리의\_이름** 의 형태로 사용한다.  
     
   - 여러 개의 디렉토리를 만들 때에는 만들어질 디렉토리의 이름을 공백으로 구분하여 나열  
    –p /{ , } 의 형태로 하위 폴더를 함께 생성한다
3. **Ls**:특정 디렉토리의 파일 상세 내용을 출력한다. 아래와 같은 옵션이 붙을 수 있다.   
   

|  |  |  |
| --- | --- | --- |
| **-l** |  | 각 파일/디렉토리에 대한 상세한 설명을 테이블 형태로 출력한다. |
| **-a** | --all | 숨김 파일 또는 숨김 디렉토리를 표시한다. 단, . 나 .. 도 같이 표시되는 단점이 있다. |
| -A | --almost-all | -a와 같지만 .와 ..를 표시하지 않는다. |
| -r | --reverse | 결과를 [내림차순](https://namu.wiki/w/%EB%82%B4%EB%A6%BC%EC%B0%A8%EC%88%9C)으로 표시한다. |
| -h | --human-readable | 파일 사이즈를 사람이 읽을 수 있는 단위로 표시한다. |
| -o |  | -l과 기능이 똑같지만, 유저그룹 관련 정보를 표시하지 않는다. |
| -1 |  | 파일이나 디렉토리를 한 줄에 하나씩 출력한다. |
| -S |  | 크기 순으로 출력한다. |
| -t |  | 수정일 순으로 출력한다. |

1. **Cd**: change directory의 약자로, 다른 디렉토리로 이동할 때 사용하는 명령어이다.  
   이전의 디렉토리로 이동하고자 한다면 cd 뒤에 ‘,,’ 을 입력한다.    
   이 경우 상대경로가 활용되는데, 이는 현재 위치를 기준으로 상대적인 파일 경로로 이동하는 것이다.  
   ‘/’를 활용해 위치를 이동한다면 절대경로기준으로 이동하는 것이고, ‘..’의 경우 상대경로 기준이다. 절대 경로는 리눅스의 최상위 디렉토리인 루트 디렉토리를 기준으로 특정 디렉토리의 경로를 지칭하는 것이고,  **pwd**를 입력했을 때 나타나는 경로가 절대 경로이다. 반면, 상대 경로는 현재 위치를 기준으로 타겟 디렉토리의 경로를 지칭한다.
2. **Pwd**: print working directory의 약자로, 현재 위치하고 있는 경로를 출력하는 명령어이다.
3. **rmdir**: remove directory 의 약자로, 특정 디렉토리를 삭제할 때 사용된다. Mkdir과 반대로 활용된다. Mkdir과 비슷하게 활용되므로 같은 방식으로 여러 개를 한번에 삭제할 수도 있다. 그러나 내용물이 비어있는 디렉토리만을 삭제할 수 있어 , 내용물이 존재하는 디렉토리를 삭제하기 위해서는 rm 명령어를 활용해야 한다.
4. **Cat**: concatenate 의 약자로, 여러 파일들의 내용을 연결하여 출력한다. 또한 파일을 만들거나 내용물을 볼 수 있기에 많이 활용된다.   
   $ cat > file.txt : 와 같은 방식으로 해당 위치에 파일을 새로 만들 수도 있다.   
   위 사진은 cat으로 txt 파일을 만드는 예시이다.  
   $ cat filename         : 파일의 내용을 모두 보여준다  
   $ head -n filename : n줄 만큼 위세서부터 보여준다  
   $ tail -n filename     : n줄 만큼 아래에서부터 보여준다
5. **Cp**:  copy의 약자로, 파일을 다른 위치에 복사하고자 할 때에 사용한다.   
   $ cp index.first index.second  
        : index.first 파일을 index.second 란 이름으로 복사한다.  
   $ cp /home/test/\*.\*  .  
        : test 디렉토리내의 모든 파일을 현재 위치한 디렉토리로 복사.  
   위와 같은 형태로 작성한다.
6. **rm** : remove의 약자로, 파일을 삭제할 때 사용하는 명령어이다.   
   rm 명령어에는 -rf 옵션을 붙일 수 있으며. 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 바로 삭제하라는 의미가 된다. 따라서, 내용물이 존재하는 디렉토리를 삭제할 때에 rm -rf를 사용합니다. 이떄엔 휴지통을 거치지 않고 바로 시스템에서 삭제가 진행되므로 복구가 불가능하다

**10) tree**

tree를 설치하면 터미널에서 가지 형태로 파일구조를 가시성 좋게 확인할 수 있다. 현재 해당 위치에서 하위에 있는 폴더들의 위치관계와, 해당 폴더 내부에 들어있는 파일의 이름까지 확인할 수 있다. 따라서 찾고자하는 파일이 해당 위치에 있는지 확인하거나 찾는 파일의 경로를 확인하고 싶을 때 주로 활용한다.

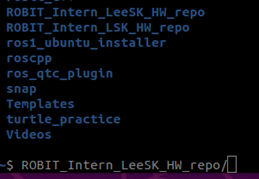
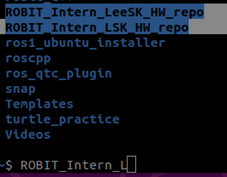
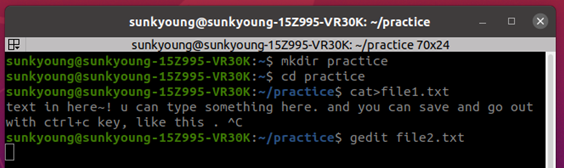
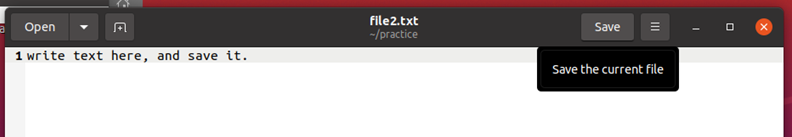


위사진은 위에 예제로 활용된 cat과 아래에 예제로 활용된 gedit으로 만들어진 파일을 tree를 통해 가시적으로 본 결과이다.

**<리눅스 특수문자 정리>**

|  |  |
| --- | --- |
| **문자** | **설명** |
| ~ | 홈 디렉토리 |
| - | 이전 작업 디렉토리 롤백 |
| . | 현재 디렉토리 |
| .. | 상위 디렉토리 |
| **히스토리 기호** |  |
| ! | history 명령어에는 고유의 식별번호가 존재하는데, 느낌표+식별번호를 입력하면 해당 번호의 명령어가 실행 |
| !! | 바로 직전에 사용한 명령어를 실행 |
| !-번호 | 마지막 명령어로부터 뒤로 번호만큼 째 명령어를 재호출 한다는 의미 !-2 는 마지막에서 두번째 명령어 호출 |
| !^ | 이전 명령어의 첫 번째 argument |
| !$ | 이전 명령어의 마지막 argument |
| !\* | 이전 명령어의 전체 argument |

<  **리눅스 조금 더 편리해지는 터미널 창 도구들>**

1. **↑/ ↓**  
   이전에 입력했던 내용, 혹은 그 이후 입력했던 내용을 터미널창에서 키보드로 위,아래 키를 통해 열람하고, 바로 시간을 단축하여 명령을 실행할 수 있다.  
   -터미널창을 새로 열어도 기록은 반영된다.   
   -열려있던 각 터미널 창 안에 있는 기록들은 각각 따로 기억되어, 예를들어 한번에 로스코어, 로스런 a, 로스런 b, rqt\_graph 이렇게 여는 창을 한번에 열어뒀을 때, 각각을 ^c로 나간 뒤 윗 방향 키를 입력하였을때, 각각이 로스코어, 로스런 a, 로스런 b, rqt\_graph 로 각 창에서의 기록을 바로 불러와 쉽게 작동할 수 있다.
2. **X tab**  
   x로 시작하는 모든 명령어를 볼 수 있다. 특정한 디렉토리를 열거나, 파일을 열거나, 런치를 할 때 등등에서 파일 이름이 긴 경우가 많을 수 있다. 이때마다 모든 글을 직접 작성하기보다 예를들어 sunkyoung 이라는 파일에 접근할 때 만일 sun\_kyoung과 같이 겹치는 글자가 없는 sunk 부터는 탭을 누르면 바로 sunkyoung이 자동완성으로 입력이 된다. s에서 누른다면 sun까지 자동으로 완성되었다.  
     
   위 예시 사진 첫번쨰는 RO를 치고 tab 키를 누른 경우이며 오른쪽은 이어 e를 치고 탭을 눌렀을 때의 사진이다. 이처럼 편리하게 활용이 가능하다.  
    또한 해당 기능은, 찾는 현재 파일경로에서 하위에 위치하여있거나, 혹은 부르고자하는 양식에 맞는 노드 이름을 완성해준다.
3. **Gedit**   
   cat>파일 이름 과 마찬가지로, 새로운 파일을 생성하고, 해당 파일을 열어주어 편집할 수 있도록 한다.  
   cat의 경우, 터미널창에서 해당 파일을 열어 맨 위에부터 열람하거나, 끝에서부터 열람하거나, 내용전체를 불러와 읽고 편집하거나 내용을 추가하거나, 등등의 작업을 할 수 있었다.   
   gedit 의 경우 파일을 생성한 뒤 터미널이 아닌 해당 파일을 직접 열어주어 해당 파일 내부에서 내용을 편집할 수 있다. 파일을 저장하고 닫으면 파일이 새로 만들어져 작성한 내용 또한 잘 저장된 것을 확인할 수 있다.  
   gedit으로 파일을 하나 더 만들었다.  
   위와 같은 파일이 열리게 되고 해당 문서 내용을 편집할 수 있다.
4. **Echo**  
   현재 확인하고자 하는 것을 echo로 확인할 수 있다.  
   주로 노드끼리 통신을 할 때 그 과정에서 각 노드에서 어떤 정보를 보내고 있는가 등을 확인할 때, echo를 통해 관심있게 보고자 하는 토픽 명과 그 안의 노드가 어떻게 실행되고있는지 값이 어떻게 전달되고 있는지를 echo를 뒤에 적어줌으로서 확인할 수 있다.
5. **Screen**  
   --screen 을 뒤에 붙여주면 rosrun을 할 때 해당 통신과정을 터미널 창에서 확인할 수 있다.

**[2. Git]**

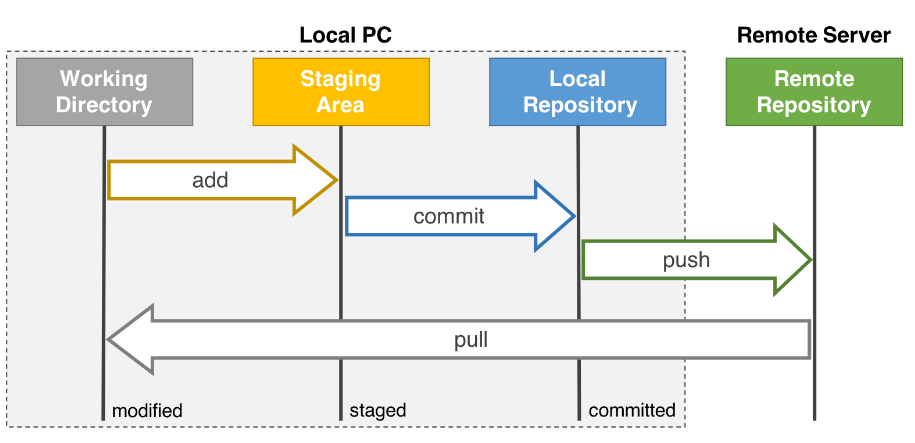
1. **Git이란?**

   
 : 분산 버전관리 시스템으로, 컴퓨터 파일의 변경사항을 추적하고 여러명의 사용자들 간에 파일에 대한 작업을 조율하는데 사용된다. 즉, 여러 개발자가 하나의 소프트웨어 개발에 참여할 때 공동으로 소스코드를 편리하게 관리할 수 있어진다.

1. **장점**  
    : 인터넷 연결이 되지 않은 곳에서도 개발을 진행 할 수 있으며, 분산버전관리이기 때문에 중앙 저장소가 삭제되어도 원상복구가 가능하다. 또한 각각의 개발자가 브랜치에서 개발한 뒤 본 프로그램에 합치는 merge방식을 통해 병렬적인 개발이 가능하여 보다 효율적인 개발이 가능하다. 따라서 하나의 프로그램을 여러명이 분업하여 만들때, 동시업로드로 인한 충돌이 적고, 작업은 로컬에서 진행하며 업로드만 네트워크를 활용하기 때문에 속도 또한 빠르다. 히스토리 관리 기능이 잘 갖추어져 있어 히스토리 관리가 용이하기도 하다.   
    즉, 정리하면

* **버전관리**: 문서를 수정할 때 언제, 무엇을 , 어떻게 수정 혹은 추가 및 변경하였는지 편리하고 구체적으로 기록하여 다른 사용자도 함께 프로젝트 버전관리에 유용하게 만들어진 관리 시스템이다.
* **백업** : 또한 현재 컴퓨터에 있는 자료를 올려둠으로서 원격 저장소이자 온라인 저장소로 백업공간을 제공하여 유용하게 파일을 올리고, 또한 저장소에 올라가있는 내용을 쉽게 빼올 수 있다.
* **공동작업** : 깃허브와 같이 협업과정에서 일어날 수 있는 파일 충돌문제 와 같이, 같은 부분을 서로 다른 사람들이 수정을 하였을 때, 자신의 개인 로컬 저장소 속의 파일과 현재 작업되고있는 깃허브 내의 파일이 다를 때, 깃허브 내의 파일들을 pull하여 개인 로컬저장소와 환경이 일치하도록 만들어 준 뒤 파일을 업로드 가능하도록 함으로서 공동 작업들에서 발생할 수 있는 문제들을 중간에서 처리해주는 역할을 한다.

1. **구조**



**Working directory** : 사용자 PC 내의 디렉터리로 현재 작업중인 깃 프로젝트 파일이 들어있는 디렉터리이다. 즉, 현재 작업하고 있는 파일들이 위치하는 사용자 디렉토리이다.

**Staging area** : 깃에 등록할 커밋 파일들이 올라가는 장소로, 커밋할 변경점들이 대기하는 곳이다. Git add 명령어를 활용하여 working directory의 변경점들을 staging area에 올리게 된다.

**Local Repository** : 커밋들이 스냅샷으로 기록된 곳으로, ‘git commit’ 명령어를 활용하여 Staging area 에 있는 변경점들을 실제 스냅샷으로 묶어 로컬 레포지토리에 올릴 수 있게 된다. 즉 커밋에 담긴 내용들이 버전의 업데이트를 기록하는 변경사항들이 되어 버전 구분점으로 활용되는 중요한 개념이다. 즉 로컬 레포지토리는 로컬 깃 프로젝트의 메타데이터와 데이터 정보가 저장되는 영역을 말한다.

**Reomote repository** : 깃의 원격 서버에 최종적으로 파일이 저장되어 보관되는 공간이다. 이곳에서는 다른 사용자에게도 공개된 레포지토리라면 함께 내용을 공유하고 확인할 수 있고, 이곳 원격저장소에서 로컬로도 내용물을 가져올 수 있다. 또한 커밋단위로 해당 프로젝트의 버전에 따라 프로젝트를 내려받는 것 또한 가능하다. 외부 저장소이기 떄문에 로컬환경속의 파일이 손상되더라도 외부 저장소에 저장된 기록은 유지되어 다시 내려받아 활용하기에도 편리하다.

**3) 기본 용어**

**Repository** : 저장소는 히스토리, 태그, 소스의 가지치기 혹은 브랜치에 따라 버전을 저장하며 작업자가 모든 히스토리를 확인할 수 있다. 프로젝트가 실질적으로 위치하는 디렉터리나 저장 공간을 의미한다. 해당 저장소 안에 코드, 텍스트, 이미지 등 다양한 자료를 저장하고 명명할 수 있다.

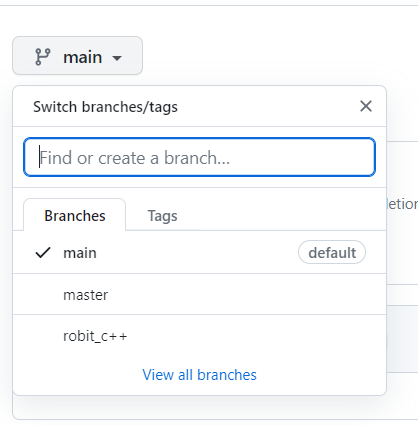
**Command line** : 깃 명령어를 입력할 떄 활용하는 컴퓨터 프로그램으로, 터미널이라고도 한다. 프롬프트라고도 알려진 텍스트 기반 명령어를 입력하여 사용자가 원하는 기능을 수행하게 된다.

**Commit** : 커밋은 쉽게 말한 개념으로는 저장소의 스냅샷을 찍어 프로젝트를 이전의 상태로 재평가하거나 복원할 수 있는 체크포인트를 만드는 일이다. 보통 큰 작업일수록 커밋 규칙에 따라 해당파일이 어떻게 변화되었는지를 기입하며 서로 다른 사용자끼리도 해당 변화가 어떤 변화인지 쉽게 파악할 수 있도록 전달하는 역할을 한다.

아래는 커밋 규칙이다.

**커밋 메시지의 7가지 규칙**

* 제목과 본문을 빈 행으로 구분한다.
* 제목은 50글자 이내로 제한한다.
* 제목의 첫 글자는 대문자로 작성한다.
* 제목 끝에는 마침표를 넣지 않는다.
* 제목은 명령문으로 사용하며 과거형을 사용하지 않는다.
* 본문의 각 행은 72글자 내로 제한한다.

**Branch :**  


여러 명이 작업할 때, 작업자들은 일반적으로 메인프로젝트의 브랜치를 가져와 자신이 변경하고자 하는 변경점을 반영한 버전을 만들어 메인 디렉토리 마스터에 자신의 변경점이 반영된 브랜치를 다시 병합하게 된다.

1. **Git 설치 및 사용**

**-초기설정-**

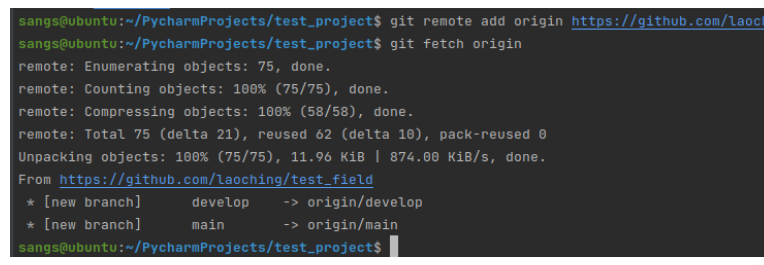
1) 깃 설치  


위 명령어를 실행하여 깃을 설치한다.

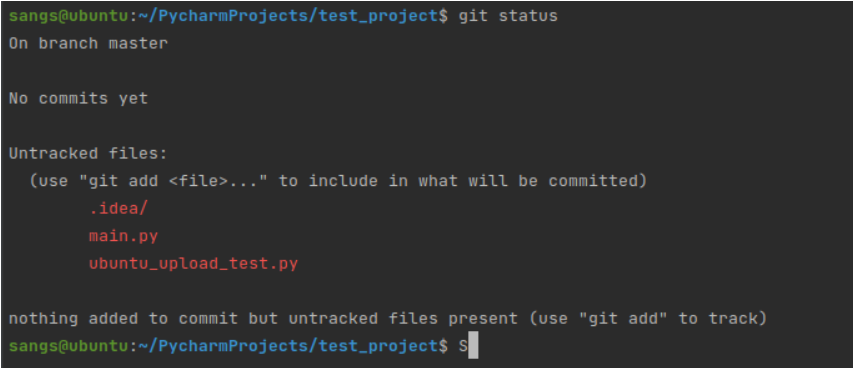
2) 이후 업로드 디렉토리를 정하고 해당 경로에서 사용자의 이름과 이메일을 설정해줄수 있다.

[git config --global user.name ~~]

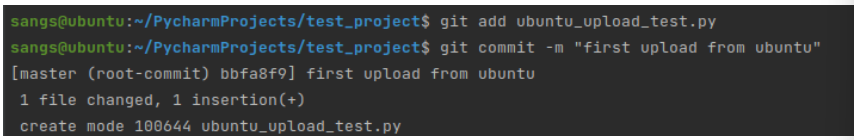
[git config --global user.email ~~]

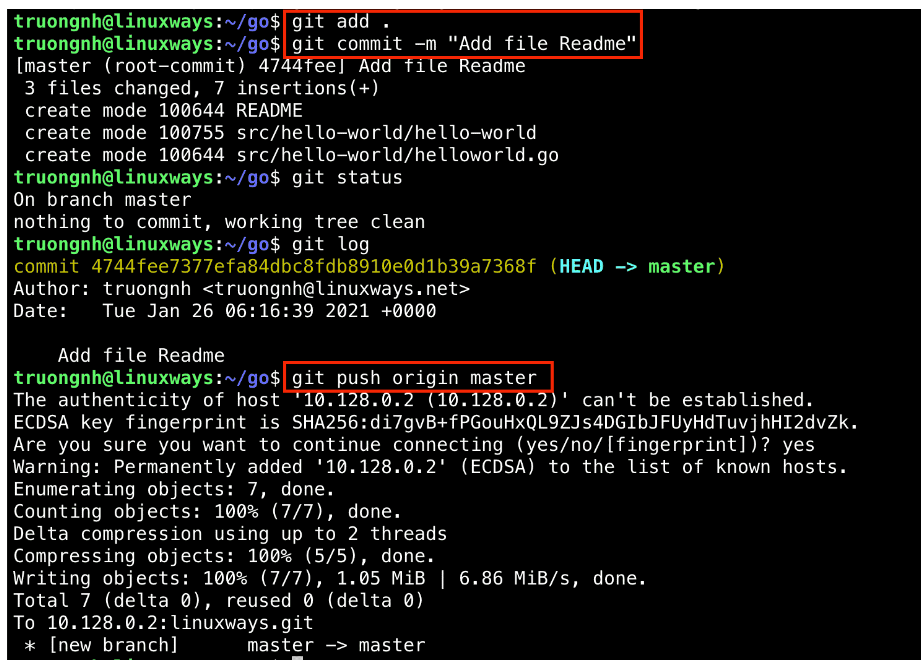
1. 이후 git init으로 저장소를 생성한다.’  
   
2. Commit 할 레포지토리를 지정한다.  
   [git remote add origin (https~저장을 원하는 저장소 경로)]
3. Get fetch origin으로 저장소를 확인한다.  
   [git fetch origin]  
     
   정상적으로 연동이 되었다면 위의 화면과 같이 브랜치들과 현재 레포지토리의 상태가 뜨게 된다.

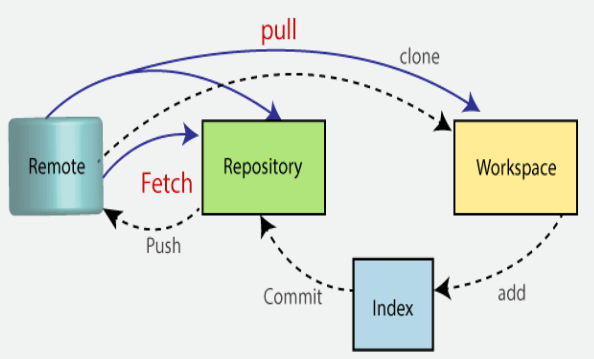
**-파일 업로드 과정-**1) **git status**  
현재 저장소의 상태를 나타내 보여준다. 아직 커밋되지 않은 채 업로드되지않은 파일들이 빨갛게 표시된다.



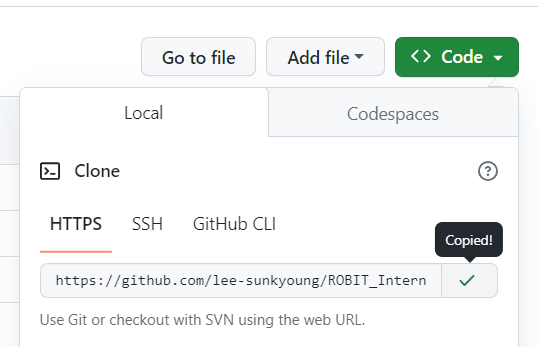
위 화면의 빨간 파일들은 즉 커밋후 업로드 가능한 파일들이다.

**2)** **git add 파일이름**  
레포지토리 파일 경로에서 업로드하고자 하는 파일을 git add 를 해준다.  
  
**3) git commit –m “커밋메시지”**커밋메시지로 전달하고자 하는 메시지를 작성한다.   
이후 이는 파일이 올라간 후 다른 사용자들도 확인하며 변경점을 아는 데에 활용된다.   
다음과 같이 해당 파일 내부에서 올라가있는 기존 파일과 달라진 점이 체크가 되어 나타난다.

**4) git push**  
[git push origin 올리고자하는 브랜치]  
의 형태로 작성하면 된다.   
성공적으로 업로드된다면 다음과 같은 화면이 뜨게 된다. 이후 깃허브에 들어가 파일을 확인하면 잘 업로드된 것을 확인할 수 있다.

**-파일 불러오기-  
  
1) 서버 소스 파일 업데이트 (fetch, pull**)  
  
fetch 와 pull은 모두 소스를 불러온다.그러나 그 방식에 있어, fetch의 경우 데이터를 가져오지만 로컬에 병합하지 않는다. Pull의 경우 병합을 진행하기 때문에 가능하다면 fetch를 통해 변경사항을 확인한 후 pull을 하는 것을 추천한다.  **- git fetch**  
fetch 의 경우 마찬가지로 원격저장소에서 로컬저장소로 소스를 가져오지만, 가져온소스를 현재 로컬의 소스에 merge를 하지 않는다.

**-  git pull [원격저장소명] [브랜치명]**   
   
git pull은 로컬에 존재하는 Git 저장소에 원격 저장소의 최신 변경사항을 가져오고 병합하는 작업을 한다. 이 명령은 원격 저장소에 새로운 커밋이 추가되었을 때, 그 변경사항을 현재 로컬 저장소에 반영하기 위해 사용된다.   
따라서 이 명령어를 실행하면 해당 원격저장소 안에 있는 해당 브랜치의 내용을 개인 로컬 저장소에 업데이트할 수 있다. 원격저장소의 정보가 바뀌었을 경우 해당 저장소에 Push를 하기 전, pull을 하는 과정이 꼭 필요하다. 정리하자면 Git pull은 git fetch를 하여 불러온 정보에 merge하는것을 합친 것과 같다.   
  
  
**2) 주소를 통해 소스 가져오기  
-git clone [https~불러오고자하는 저장소 주소]**



파일을 저장할 경로에서 불러오고자 하는 레포지토리의 부분에서 clone 하기위한 주소를 복사해 git clone 주소를 작성하여 명령어를 입력하면 해당 경로에 clone한 내용이 저장되게 된다.

이는 원격 저장소의 전체 복사본을 로컬에 새로 만드는 것과 같은 일이기 때문에, 새로운 디렉토리가 생성되며 그 안에 원격 저장소의 모든 파일과 폴더, 그리고 모든 커밋 히스토리가 포함되게 된다. 따라서 이 명령은 주로 처음 레포지토리를 불러오는것을 시작할 때 사용되며, 그 이후에 수정과정 등에서 일반적으로 자주 사용되지 않는다.  
  
**-원격저장소 변경-**

기본적으로 파일은 push를 했을 때 처음 원격저장소로 설정한 곳에 들어가게 된다. 그렇기 때문에 올리고자하는 리포지토리를 바꾸기 위해서는 리모트 변경이 필요하다.

다음은 기존 리포지토리를 제거하고 새로운 리포지토리를 추가하는 명령어이다.

[**기존 리포지토리 원격제거**](https://gist.github.com/480/4681b67d2a906db8c6c1321cc678f05f#%EA%B8%B0%EC%A1%B4-%EB%A6%AC%ED%8F%AC%EC%A7%80%ED%86%A0%EB%A6%AC-remote-%EC%A0%9C%EA%B1%B0)

git remote remove origin

[**새 리포지토리 원격 추가**](https://gist.github.com/480/4681b67d2a906db8c6c1321cc678f05f#%EC%83%88-%EB%A6%AC%ED%8F%AC%EC%A7%80%ED%86%A0%EB%A6%AC-remote-%EC%B6%94%EA%B0%80)

git remote add origin (https~깃허브/계정/리포지토리)

해당 명령어를 통해 리포지토리 변경이 가능하다.

1. **Git 주요 명령어**

**$ git init**

깃 저장소를 초기화한다. 저장소나 디렉토리 안에서 이 명령을 실행하기 전까지는 그냥 일반 폴더이지만, 이것을 입력한 이후 추가적인 깃 명령어들을 줄 수 있다.

**$ git config**

“configure”의 준말, 처음에 깃을 설정할 때 가장 유용하다. 이 config로 사용자 이름, 이메일 등을 저장해두기도 한다.

**$ git help**

명령어를 잊어버렸을 때 커맨드 라인에 이걸 타이핑하면 21개의 가장 많이 사용하는 깃 명령어들이 나타난다. 좀 더 자세하게 명령어를 확인하기 위해서는 $ git help init이나 다른 용어를 타이핑하여 특정 깃 명령어를 사용하고 설정하는 법을 이해할 수도 있다. 도움말처럼 사용하는 양식이 나타난다.

**$ git status**

저장소 상태를 체크하는 역할을 한다. 현재 어떤 파일이 저장소 안에 있는지, 커밋이 필요한 변경사항이 있는지, 현재 저장소의 어떤 브랜치에서 작업하고 있는지 등을 볼 수 있다. 따라서 커밋과 저장하며 제대로 파일이 add가 되었는지 확인하거나, 찾고자하는 파일이나 올리고자하는 파일이 저장소 내부에 위치하고 있는지 확인하는데 유용하다.

**$ git add**

이 명령이 저장소에 새 파일들을 바로 추가하는 작업은 아니다. 대신, 깃이 새 파일들을 지켜보게 한다. 파일을 추가하면, 깃의 저장소 “스냅샷”에 포함된다. 변경점들이 커밋할 변경점대기상태에 들어가게 된다.

**$ git commit**

깃의 특징을 가장 잘 나타내는 중요한 명령어이다. 어떤 변경사항이라도 만든 후, 저장소의 “스냅샷”을 찍기 위해 이것을 입력한다. 보통 $ git commit -m “Message” 형식으로 사용한다. -m은 명령어의 그다음 부분을 메시지로 읽어야 한다는 것을 말한다. 이는 주로 깃 커밋 규칙에 따라 작성하게 된다.

**$ git branch**

여러 협업자와 작업하며 자신만의 변경점을 반영하고 싶을 때 주로 활용한다. 이 명령어는 새로운 브랜치를 만들어, 자신만의 변경사항과 화일 추가 등의 커밋 타임라인을 만든다. 사용할 때에는 명령어+ 제목의 형태로 작성한다. 새로운 브랜치를 “sunkyoung”으로 부르고 싶으면, git branch sunkyoung으로 프로젝트 내의 자신의 작업 변경점업로드 공간인 브랜치를 만들 수 있다.

**$ git checkout**

글자 그대로, 현재 위치하고 있지 않은 저장소를 “체크아웃”할 수 있다. 이것은 체크하길 원하는 저장소로 옮겨가게 해주는 탐색 명령이다. master 브랜치를 들여다보고 싶으면, git checkout master를 사용할 수 있고, git checkout “브랜치 이름”으로 또 다른 브랜치를 들여다볼 수 있다.

**$ git merge**

브랜치에서 작업을 끝내면, 모든 협업자가 볼 수 있는 master 브랜치로 병합할 수 있다.

git merge “브랜치이름”은 해당 브랜치이름을 가진 브랜치에서 만든 모든 변경사항을 master로 추가하여 반영하게 하는 명령어이다.

**$ git push**

로컬 컴퓨터에서 작업하고 당신의 커밋을 깃허브에서 온라인으로도 볼 수 있기를 원한다면, 이 명령어로 깃허브에 변경사항을 push한다.

**$ git pull**

로컬 컴퓨터에서 작업할 때, 작업하고 있는 저장소의 최신 버전을 원하면, 이 명령어로 깃허브로부터 변경사항을 다운로드한다(pull). 저장소의 변경점을 다운로드 하지 않고 푸시를 먼저 하려고 하면 오류가 뜨게 되므로 먼저 원격저장소의 환경과 로컬 컴퓨터의 상황을 동일하게 만든 뒤 작업하여야한다.