<ROBIT Linux & Git 보고서>

18기 지능팀 예비단원 주성민

1. Linux

1) 정의: Linux는 1991년 Linus Torvals가 개발한 운영체제이다. Linux는 Unix 운영체제를 기반으로 만들어진 운영체제로 유닉스 클론 운영체제라 할 수 있다. 다중 사용자, 다중 작업, 다중 스레드를 지원하는 네트워크 운영체제를 의미한다. 또한 unix가 통신 네트워크를 지향해 설계된 것처럼, Linux 역시 서버로 작동하는데 최적화 되어있다. Linux는 자유 소프트 라이선스로 누구나 소스코드를 활용, 수정 및 재배포가 가능해서 지속적인 업데이트가 이루어진다.

* Linux와 Ubuntu 차이: Linux에는 다양한 배포판이 존재한다. 예를 들어 Red Hat, CentOS, Debian 등이 있다. 우분투 또한 리눅스 배포판이다.

[공통점]

- 둘 다 무료이며 오픈소스이다.

- 다중 사용자를 지원하며 멀티 태스킹과 멀티 쓰레딩이 가능하다

- 안전하고 안정적이다

[차이점]

- 리눅스는 리눅스 커널 기반으로 구축된 무료 오픈 소스 소프트웨어 운영체제인 반면 우분투는 데비안의 무료 오픈 소스 운영체제 및 리눅스의 배포판이다.

- 리눅스가 핵심 운영체제이고, 우분투는 리눅스 배포본인 운영체제이다.

- 리눅스는 개인용 컴퓨터, 게임 콘솔, 임베디드 시스템, 데스크톱 및 서버에 사용되는 반면 우분투는 개인용 컴퓨터, 서버, 클라우드 컴퓨팅 및 IoT에 사용된다.

2) 구조: 리눅스는 크게 커널, 셸, 디렉토리로 3가지로 구성되어있다.

- **커널(kernel)**: 커널은 운영체제의 핵심으로 메모리 관리, 프로세스 관리, 장치 관리 등 컴퓨터의 모든 자원을 초기화하고 제어하는 기능을 수행한다.

- **셸(shell)**: 사용자가 입력한 문장을 읽어 요청을 실행하며 커널이 명령어를 해석해 결과를 수행한 후 결과를 다른 프로그램이나 커널로 전송한다. 즉 사용자와 커널의 중간다리 역할을 수행한다. 리눅스는 bash 셸을 기본으로 사용한다. 셸은 커널에서 분리된 별도의 프로그램으로 다양한 종류의 셸이 존재하며 현재까지도 지속적으로 개발되고 있다.

* shell 종류

- **bourne Shell(sh)**: 기본적으로 설치가 되어있는 최초의 쉘

- **bourne-Again Shell(bash)**: GNU 프로젝트 일화로 개발됨, 리눅스에서 지원되는 기본 셸로 사용자가 계정을 생성할 때 특별히 지정하지 않으면 기본적으로 지정된다

- **korn shell**: sh와 호환되며 cshell의 많은 기능을 포함한다. 유닉스에서 가장 많이 사용되는 셸로 명령행 편집기능을 한다.

- 터미널

컴퓨터와 사용자간 서로 소통시켜주는 인터페이스

터미널은 크게 텍스트 기반, 그래픽 기반 두가지를 기반하는 용어지만 리눅스 터미널은 텍스트 기반을 의미한다.

터미널 쉘 차이: 터미널은 쉘을 실행시키는 프로그램이고, bash, zsh를 실행시킬 수 있다.

터미널 단축키: ctrl + Alt + T

<터미널 설치 방법>

$ sudo apt-get update

$ sudo apt-get upgrade

$ sudo apt-get install terminator

<terminator 설정 방법>

터미널 창 우클릭 -> preferences 클릭 -> profiles – scrolling -> infinitescrollback활성화

=> 터미널 줄바꿈 횟수가 제한적이었지만 터미네이터 설정을 통해 줄바꿈 횟수를 무제한으로 늘려주면서 디버깅에 용이해짐

<터미널 분할 단축키>

Ctrl + shift + E: 좌우 분할

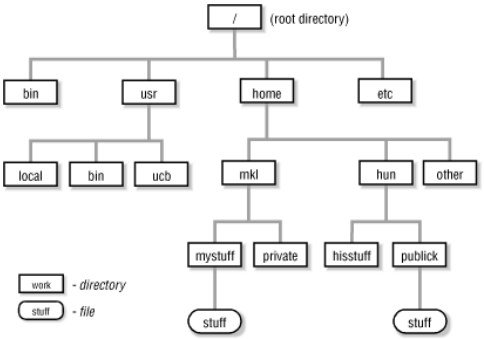
Ctrl + shift + O: 상하 분할

Ctrl + shift + W: 창 닫기

Alt + 화살표: 창 이동

- **디렉토리(directory)**: 쉽게 말해 파일 저장소로 리눅스 디렉토리는 최상위 디렉토리를 기준으로 하위 디렉토리들이 존재하는 계층적 트리 구조로 되어있다.

* 파일 시스템 계층 구조



리눅스의 디렉토리 구조는 전체적으로 tree 구조이며 명령어의 성격과 내용 및 사용권한 등에 따라 디렉토리로 구분된다.

~: 홈 디렉토리, 터미널 구동 시 최초의 위치

/: FHS 의 가장 최상단 디렉토리이며 Root 디렉토리라고도 함, 시스템의 근간을 이루는 가장 중요한 디렉토리로 파티션을 설정 시 반드시 존재 해야하며, 절대 경로의 기준이 되는 디렉토리

/bin: 리눅스의 기본적인 명령어가 저장된 디렉토리로 시스템을 운영하는데 가장 기본적인 명령어들이 모여있다.

/home: 사용자 홈 디렉토리, 일반 사용자의 홈 디렉토리가 만들어지는 곳

/boot: 부트 설정 파일과 lilo 를 제외한 부트 관련 모든 파일을 모아놓은 디렉토리

* lilo 란, 리눅스 부트 로더이다. 파일 시스템에 의존하지 않으며 플로피 디스크와 하드 디스크로부터 운영체제를 시동할 수 있다.

/dev: 시스템의 모든 디바이스를 접근할 수 있는 파일들을 모아놓은 디렉토리

/etc: 시스템 환경 설정 파일과 부팅 관련 스크립트 파일들을 저장하는 디렉토리

/usr: 일반 사용자들을 위한 대부분의 프로그램 라이브러리 파일이 위치한다. /user/bin: 일반 사용자들이 사용 가능한 명령어 파일들이 존재하는 디렉토리이고, /user/local: 새로운 프로그램들이 설치되는 공간이다.

/lib: 공유 라이브러리 디렉토리로 커널 모듈 파일들과 프로그램 실행을 지원해주는 라이브러리를 저장한다.

* 디렉토리 경로

- **Absolute Path**: 이름 그대로 절대적인 경로, 완전한 경로를 의미하고 Root 디렉토리로부터 시작하는 경로를 뜻한다. 현재 나의 위치와 상관없이 항상 정확한 경로로 전달한다.

- **Relative Path**: 이름 그대로 상대적인 경로를 의미하고 현재 내 위치를 기반으로 움직인다. .과 ..이 두가지 심볼이 중요하다. .은 현재 디렉토리, ..는 상위 디렉토리를 의미한다.

3) 설정

Configs: Linux 에서는 설정을 주로 파일을 통해서 진행한다. 그리고 여러 설정파일들이 존재한다. 그 중에서 가장 중요한 파일은 shell 설정파일이다. 각 shell 마다 고유 설정파일들이 존재한다. Zsh 는 .zshrc, bash 는 .bashrc 가 존재한다. 이러한 설정 파일도 중요하지만, PATH 즉 환경 변수 설정이 가장 중요하다.

환경 변수: 운영체제 즉 윈도우, linux 등 모두 path 환경 변수가 존재한다. 먼저 변수란 어떠한 값을 저장하고 있는 것을 의미한다. 그 앞에 환경이 붙은 것으로 환경이란 여기서 shell 을 의미한다. 그렇기에 환경 변수는 shell 의 설정 값을 가지는 변수라고 할 수 있다. 이 환경변수들은 shell 이 가동되는 동안 계속 존재하며 사용된다.

linux 에서는 환경 변수가 몇가지 존재한다.

- **Home**: 유저의 home 디렉토리 경로를 저장한 환경 변수

- **USER**: 유저의 아이디 값을 저장한 환경 변수

- **PATH**: PATH 값을 저장한 환경 변수, 이는 명령어들을 찾는 경로를 저장해 놓은 환경변수이고 새로운 패키지나 시스템을 설정할 때 PATH 가 설정이 안되어 있으면 실행이 제대로 안되는 경우가 존재한다.

4) Package Manager

: package manager 는 패키지를 다루는 작업을 편리, 안전하게 수행하기 위해 사용되는 툴이다. 각 OS 마다 사용하는 package manager 가 다르다 ubuntu 에서는 apt 나 apt-get 을 사용한다.

- apt 란? Advanced package tool 의 줄임말로 리눅스게열 패키지 관리 명령어 도구이다. Apt 와 apt-get 의 차이점은 apt-get 의 옵션이 많아서 자주 사용하는 옵션들을 추출해 편의성을 늘린게 apt 다.

5) 명령어 모음

[shell의 기본 명령어]

cd	Change directory의 줄임말로 현재 디렉토리를 변경하는데 쓰인다

ls	List directory contents에서 유래된 명령어로 해당 디렉토리 내에 있는 디렉토리 및 파일을 화면에 출력한다
mv	move의 줄임말로 파일 혹은 디렉토리로 이동한다
cp	copy의 줄임말로 파일 혹은 디렉토리를 복사할 수 있다. 디렉토리를 복사할 땐 –r 옵션을 주어야한다
cat	concatenate의 줄임말로 파일 내용 출력도 가능하며 여러 개의 파일을 하나로 만들거나 한 파일의 내용을 다른 파일로 덧붙이는 것 또한 가능하다
tail	파일의 뒷부분을 보여주는 것이다. 하위 10줄을 출력한다
nohup	셸 스크립트 파일을 데몬 형태로 실행시키는 것으로 터미널 세션이 끊겨도 실행을 멈추지 않고 동작하도록 한다
rm	Remove의 줄임말로 디렉토리나 파일을 삭제할 때 씌며 디렉토리를 삭제할 땐 r 옵션을 주어야한다

mkdir	Make a directory의 줄임말로 디렉토리를 생성시킨다. –p 옵션을 준다면 하위 디렉토리까지 한번에 생성이 가능하다
clear	터미널의 내용을 모두 지우는 명령어이다
pwd	Print working directory의 줄임말로 현재 작업중인 디렉토리 정보를 출력한다
ps	Process status의 약자로 현재 돌아가고 있는 프로세스를 확인할 수 있다

[apt 관련 명령어]

Sudo: superuser do 에서 유래, 프로그램이 확장되면서 substitute user do(다른 사용자의 권한으로 실행)로 해석

sudo apt-get update	패키지 인덱스 정보를 업데이트
sudo apt-get upgrade	설치된 패키지 업그레이드
sudo apt-get dist-upgrade	의존성 검사하며 dependency 까지 설치
sudo apt-get install <package name>	패키지 설치

sudo apt-get –reinstall install <package name>	패키지 재설치
sudo add-apt-repository <저장소이름>	저장소 추가
sudo add-aptrepository –remove<저장소이름>	저장소 제거
sudo apt-get remove <package name>	설정파일은 지우지 않고 패키지 삭제

sudo apt-get –purge remove <package name>	설정파일까지 모두 삭제
sudo apt-get source <package name>	패키지 소스 코드 다운로드
sudo apt-get build-dep <package name>	위에서 받은 코드 의존성 있게 빌드
sudo apt-cache search <package name>	패키지 검색
sudo apt-cache show <package name>	패키지 정보 확인

[VI(visual editor): 유닉스 계열에서 많이 사용되는 편집기]

실행 단축키:

- 저장 및 종료

:w	저장
:w file.txt	File.txt 파일로 저장
:w >> file.txt	File.txt 파일에 덧붙여서 저장
:q	Vi 종료
:q!	Vi 강제 종료
ZZ	저장 후 종료
:wq!	강제 저장 후 종료
:e file.txt	File.txt 파일을 불러옴

:e	현재 파일을 불러옴
:e#	바로 이전에 열었던 파일을 불러 옴

- 입력모드 전환

a	커서 위치 다음칸부터 입력	*A*	커서 행의 맨 마지막부터 입력
i	커서의 위치에 입력	*I*	커서의 행의 맨 앞에서부터 입력
o	커서의 다음행에 입력	*O*	커서의 이전 행에 입력
s	커서 위치의 한글자를 지우고 입력	*cc*	커서 위치의 한 행을 지우고 입력

- 이동

h	왼쪽으로 이동	*i*	오른쪽으로 이동
j	아래 행으로 이동	*k*	위 행으로 이동
W 또는 w	다음 단어의 첫 글자로 이동	*B 또는 b*	이전 단어의 첫 글자로 이동
E 또는 e	단어의 마지막 글자로 이동	*<CR>*	다음행 첫 글자로 이동
^	그 행의 첫 글자로 이동	*$*	그 행의 마지막 글자로 이동
+	다음 행의 첫 글자로 이동	*-*	위 행의 첫 글자로 이동
(	이전 문장의 첫 글자로 이동	*)*	다음 문장의 첫 글자로 이동
{	이전 문단으로 이동	*}*	다음 문단으로 이동
H	커서를 화면 맨 위로 이동	*z<CR>*	현재 행을 화면의 맨 위로 이동
M	커서를 중앙으로 이동	*z.*	현재 행을 화면의 중앙으로 이동
L	커서를 화면 최하단으로 이동	*z-*	현재 행의 화면의 최하단으로 이동
[n]H	커서를 위에서 n 행으로 이동	*[n]L*	커서를 아래에서 n 행으로 이동
Ctrl+u	반 화면 위로 스크롤	*Ctrl+d*	반 화면 아래로 스크롤
Ctrl+b	한 화면 위로 스크롤	*Ctrl+f*	한 화면 아래로 스크롤
gg 또는 1G	문서의 맨 처음으로 이동	*G*	문서의 맨 마지막 행으로 이동
[n]G	N 행으로 이동	*:[n]*	N 행으로 이동

- 삭제

X 또는 dl	커서 위치의 글자 삭제	*X 또는 dh*	커서 바로 앞의 글자 삭제
dw	현재 위치부터 스페이스까지 삭제	*diw*	현재 위치에 있는 단어 삭제
dd	커서가 있는 행 삭제	*[n]dd*	현재 커서부터 아래 n 번째 줄까지 삭제

dj	현재 커서와 아래 줄 삭제	*[n]dj*	현재 커서부터 아래 n+1 번째 줄까지 삭제
dk	현재 커서와 위로 n+1 번째 줄까지 삭제	*[n]dk*	현재 커서와 윗 줄 삭제
D 또는 d$	현재 커서가 있는 위치부터 행 끝까지 삭제	*d0 또는 d^*	현재 커서가 있는 위치부터 행 시작까지 삭제

- 복사, 붙여 넣기

Y 또는 yy	커서가 있는 한 행 복사
P	현재 커서에 붙여넣기
p	현재 커서위치의 앞행에 붙여넣기
[n]yy 또는 [n]Y	커서가 위치한 이후로 n 행 복사
[n]p	n 번만큼 붙여넣기 반복

[terminal 단축키]

Tap	파일 폴더 이름 자동완성
Ctrl + A	현재 작성 중인 라인에서 맨 앞으로 가기
Ctrl + E	현재 작성 중인 라인에서 맨 뒤로 가기
Ctrl + U	커서 뒤에 다 지우기
Ctrl + K	커서 앞에 다 지우기
Ctrl + W	커서 뒤에 단어 다 지우기
Ctrl + T	커서 뒤 두 글자 위치 바꾸기
ESC + T	커서 뒤 두 단어 위치 바꾸기
Ctrl + L	화면 다 지우기
Ctrl + C	실행 중인거 다 끊기
Ctrl + D	현재 SHELL 종료
OPTION +->	커서를 한 단어 앞으로 이동
OPTION +<-	커서를 한 단어 뒤로 이동
Ctrl + F	커서를 한 글자 앞으로 이동
Ctrl + B	커서를 한 글자 뒤로 이동
Ctrl + Y	마지막 명령어 다시 붙여 넣기
Ctrl + Z	실행중인거 다 종료
Ctrl + -	마지막 명령어 실행 취소

우분투 화면 녹화 시작과 종료 단축키: shift ctrl alt + r (max 30 초)

sec 초만큼 녹화 시간 증가 gsettings set org.gnome.settings-daemon.plugins.media-keys max-screencast-length (sec)

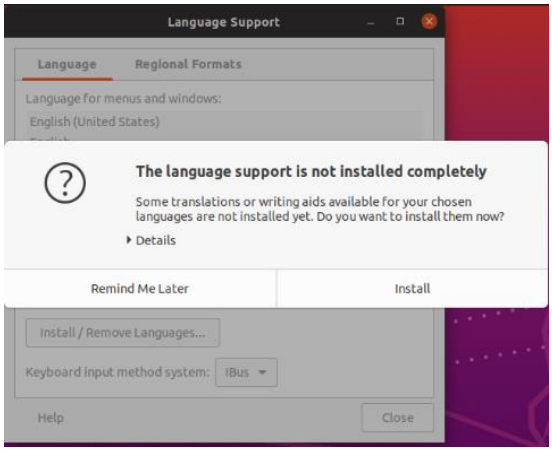
* 우분투 20.04 한글 설정

$ sudo apt-get update

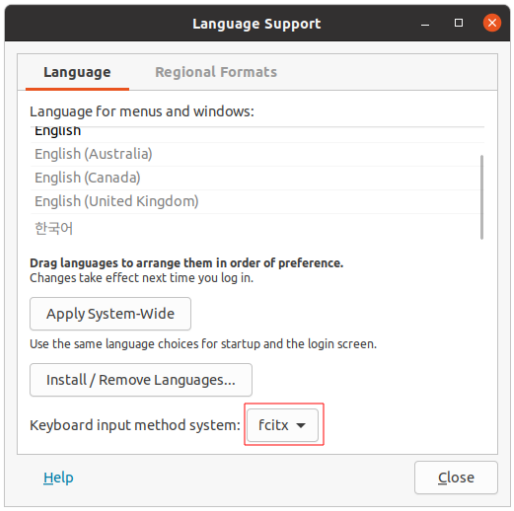
$ sudo apt-get install fcitx-hangul

- 설치가 되면 language support 설정 메뉴로 들어간다

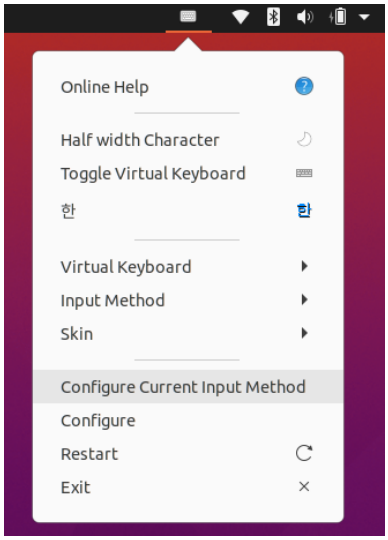
- 처음 들어가게 되면 다음과 같은 경고창이 뜨는데 install 눌러서 설치해준다



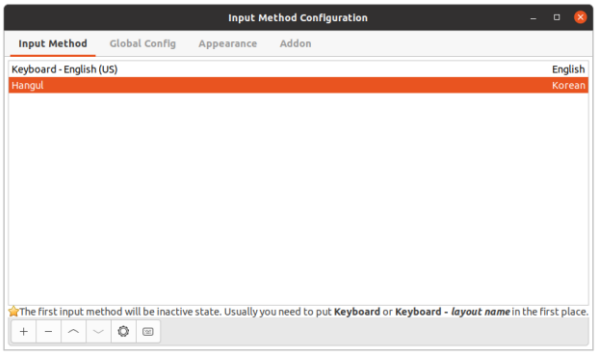
- 언어 목록 업데이트가 완료되면 다음과 같이 아래의 input method 를 fcitx 로 변경한다



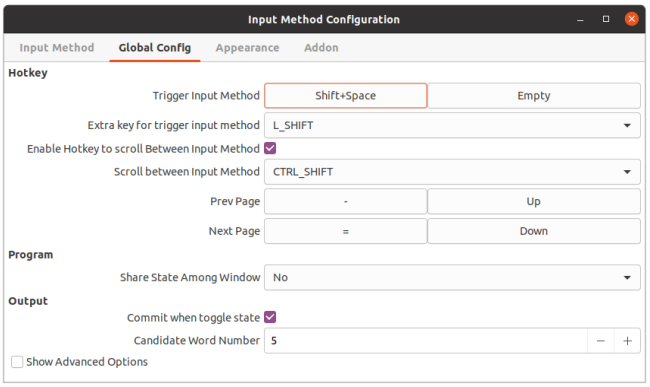
- 변경하고 나면 오른쪽 상단에 입력기 메뉴가 생기게 된다. Configure current input method 를 선택하여 fcitx 설정 창을 열어준다



- 다음과 같이 왼쪽 아래의 + 를 눌러 hangul 언어를 추가해준다. 만약 목록에 없다면 재부팅하기



- 맨 위의 단축키 설정을 통하여 한영 버튼을 변경하여 준다. 기본은 ctrl + space 지만 shift + space 로도 변경 가능



2. GIT

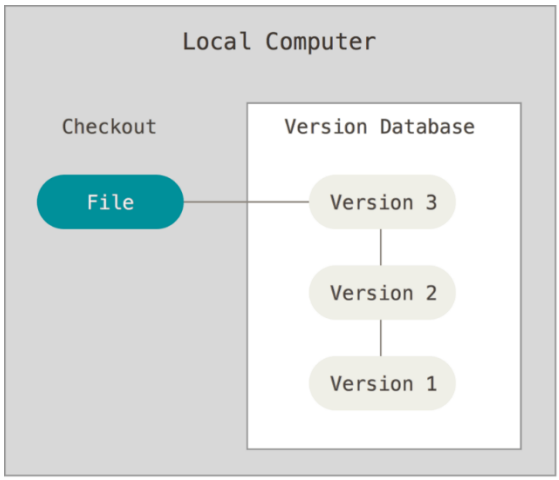
0) 정의: *버전 관리 시스템(VCS)의 하나이다. 쉽게 말해 버전을 관리할 수 있는 수단이다. 수정사항이나 업데이트 사항을 그때 그때 바로 반영할 수 있도록 하는 시스템을 일컫는다.

* 버전 관리 시스템(VCS): 파일 내 변화를 시간의 흐름에 따라 기록했다가 이후 필요한 상황에서 그 파일을 꺼내올 수 있는 시스템을 말한다. 버전 관리 시스템을 활용하면 동일한 정보에 대한 여러 버전을 관리하게 되며 버전을 통해 시간에 따른 변경 사항 및 변경자를 확인할 수 있다. 또한 이전 버전으로 돌아갈 수 있고, 다시 원래 버전으로 돌아올 수도 있으며 누가 문제를 일으켰는지도 쉽게 파악할 수 있다.

VCS 에는 3 종류가 있다.

(1) 로컬 버전 관리 시스템(Local VCS)

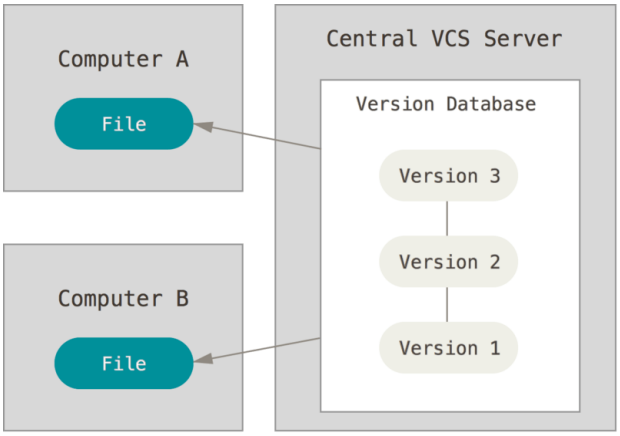
로컬 버전 관리 시스템에서는 RCS 를 활용한다. RCS 는 기본적으로 'patch set(파일에서 변경되는 부분)'을 관리한다. 이 patch set 을 통해 모든 파일을 특정한 시점으로 되돌릴 수 있다.



<Local VCS>

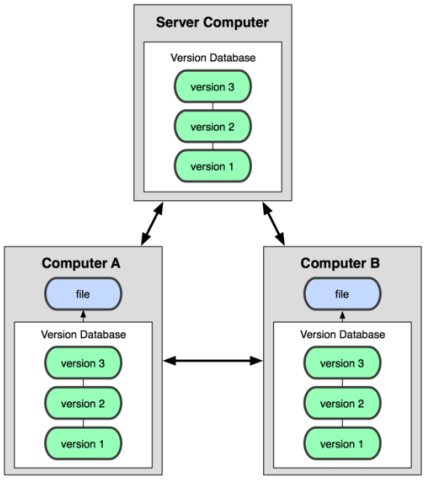
(2) 중앙 집중식 버전 관리 시스템(Centralized VCS – CVCS)

중앙 집중식 버전 관리 시스템은 프로젝트 진행 중, 다른 개발자와 협업을 진행해야할 때 사용한다. 파일 관리를 위한 서버가 별도로 존재하고, 클라이언트가 중앙 서버에서 파일을 받아서 사용한다. CVCS 를 사용하면 프로젝트에 참여한 사람들 중 누가 어떤 작업을 하는지 쉽게 확인 가능하다는 장점이 있다. 또한 모든 클라이언트의 로컬 데이터베이스를 관리하는 것보다 VCS 하나를 관리하는 것이 훨씬 쉽기 때문에 협업 시 많이 사용된다. 하지만, 중앙 서버가 다운되는 등의 문제가 발생할 경우, 그 상황 동안에는 작업이 불가능하다. 또한 하드 디스크에 문제가 발생하면 모든 히스토리를 잃을 수 있다는 단점이 존재한다.



(3) 분산 버전 관리 시스템(distributed VCS)

분산 버전 관리 시스템은 '분산'이라는 단어에 집중한다. 즉 개발자들이 독립적으로 작업한 다음에 변경 사항을 병합할 수 있기 때문이다. 분산 버전 관리 시스템에서의 클라이언트는 단순히 파일의 마지막 snapshot 을 사용하지 않는다. 저장소를 히스토리와 더불어 전부 복제하는 방식이다. 만약 서버에 문제가 생긴다면, 복제했던 것을 통해 다시 작업할 수 있다. 또한 클라이언트 중에서 아무거나 골라도 서버를 복원할 수 있다. 따라서 다양한 협업 시 주로 사용되며 깃이 이 분산 버전 관리 시스템에 속한다.

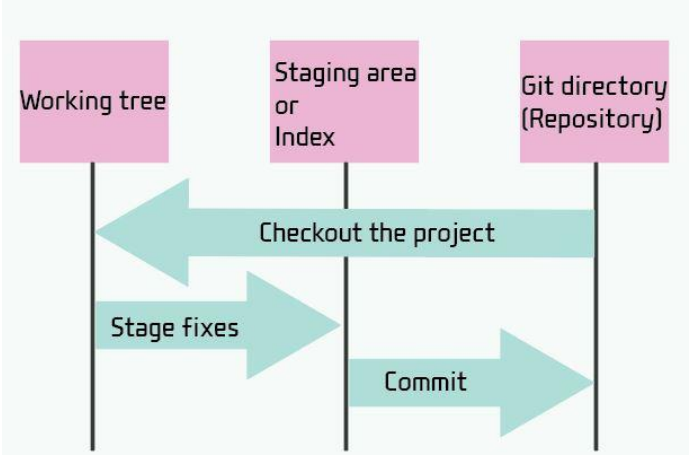


1) git 의 상태

- **committed**: 데이터가 로컬 데이터 베이스 안에 안전하게 저장되었다는 것

- **modified**: 수정한 파일을 아직 로컬 데이터 베이스에 커밋하지 않은 것

- **staged**: 현재 수정한 파일을 곧 커밋할 것이라고 표시한 상태



2) 목적: 깃을 사용함으로서 얻는 장점은 많다. 우선, 앞서 언급했던 것처럼 쉬운 버전 관리가 가능하다. 수 많은 코드들을 다루고 다른 개발자들과 협업을 진행하다 보면 잦은 업데이트 사항이 생긴다. 그럴 때마다 파일 명을 다르게 하여 새로 저장해야한다면 그 파일은 엄청나게 많아 질 것이다.

따라서 GIT 을 사용하면 이러한 번거로운 과정 없이도 다른 개발자들과 손 쉽게 코드를 주고 받을 수 있으며, 여러 명이 동시에 한 코드를 가지고 작업하는 병렬적인 작업의 진행이 가능하다.

A 라는 개발자가 코드를 작성하여 B 에게 넘겨주고, B 가 코드를 받아서 진행한 뒤 C 에게 넘겨주는 귀찮은 과정이 없어진 것이다. 또한 쉽게 이전 버전으로 이동할 수 있으며 다시 원래의 버전으로 돌아오는 것 또한 자유롭다.

GIT 은 분산 버전 관리 시스템이므로 중앙 서버가 필요 없다. 즉 인터넷이 연결되어 있지 않은 상황에서도 작업이 가능하며 도중에 이상이 생겨 저장소가 날아가버려도 쉽게 복구할 수 있다.

3) Git 과 Github 의 연관성

Git

- 로컬에서 관리되는 버전 관리 시스템

- 직접 소스 코드를 수정함으로써 버전을 관리

- 소스 코드를 효율적으로 관리할 수 있게 해주는 형상 관리 도구

Github

- Git 을 사용하는 프로젝트를 지원

- 개발자들의 버전 제어 및 협업을 위한 하나의 플랫폼

- 클라우드를 통해 관리되는 버전 관리 시스템

- 오픈 소스는 일정 부분 무료로 사용이 가능

- Git 처럼 자체적으로 구축하는 시스템이 아닌 클라우드를 빌려 쓰는 개념

➔ 간단히 Git 은 로컬에서 버전 관리 시스템을 운영하는 방식이고, github 는 깃허브 자체에서 제공해주는 클라우드 서버를 이용한다. 따라서 타인과 협업 시 Github 를 써서 오픈 소스를 공유하거나 타 개발자들과 의견을 공유할 수 있다

4) Git 의 주요 개념

- **Repository(저장소)**: 소스 코드들이 저장되어있는 물리적인 공간을 의미한다. 저장소를 통해서 작업자가 진행, 변경했던 사항들에 대해 알 수 있다. 작업을 시작할 때 원격 저장소에서 로컬 저장소로 소스 코드를 복사해서 가져오고(Clone) 이후 소스 코드를 변경한 다음 커밋(commit)을 한다. 커밋한 소스는 로컬 저장소에 저장되며 Push 하기 전에는 원격 저장소에 반영되지 않는다.

Remote: 원격 저장소 / local: 개인 컴퓨터 서버에 저장

- **Working Tree**: 우리가 사용하는 폴더를 말한다

- Index: commit 을 실행하기 전의 저장소와 Working tree 사이에 존재하는 공간을 말한다. Working tree -> index -> commit 순의 절차를 거친다.

- **Commit**: 작업 과정들에 대한 점검을 마친 뒤, 저장소에 남기는 과정을 의미한다. 각각의 커밋 단계는 의미 있는 단계이다. 따라서 커밋 로그를 남긴다. Git log 라는 명령어를 통해 커밋된 사항들에 대해 확인할 수 있다.

- **checkout**: 특정 시점이나 branch 의 소스코드로 이동하는 것을 의미한다. 이 과정을 통해 과거 여러 시점의 소스 코드로 이동할 수 있다.

- **branch**: commit 단위로 구분된 소스 코드 타임라인에서 분기해서 새로운 commit 을 쌓을 수 있는 가지를 만드는 것을 말한다. branch 에서 작업을 완료하면, Merge 작업을 수행한다.

- **merge**: Branch 와 branch 의 내용을 합치는 작업, 측 병합을 말한다. Branch 와는 다소 반대되는 개념이다. 병합 과정 중 두 branch 에서 하나의 동일한 파일에서 서로 다르게 수정한 경우 충돌이 발생하며 병합이 일시 정지 된다. 이때 충돌 부분에 대해 직접 수정하거나 Merge tool 등을 활용하여 충돌을 해결한 뒤 병합을 진행한다.

5) git 의 명령어

git init	깃 초기화. 깃 작업을 하려면 이 과정이 꼭 필요하다
git status	깃 저장소의 상태를 확인한다
git add	커밋에 의해 파일의 변경 사항을 포함한다

git commit	Git commit –m'저장명' 등과 같은 명령어로 사용한다. 커밋을 생성하고 변경 사항을 확정한다
git clone	기존 소스 코드를 다운로드 및 복제한다, 원격 저장소의 저장소를 로컬에서 이용할 수 있도록 복사해온다.
git log	나의 커밋 내역에 대해 알고 싶을 때 사용하면 현재 커밋 목록들을 확인가능하다
git checkout	브랜치에서 브랜치로 이동이 가능하다. 현재 버전에서 이전 버전의 커밋으로 이동하거나 변경 전의 브랜치로 접근이 가능하다
git checkout master	변경 전 브랜치에서 다시 현재의 브랜치로 되돌아올 수 있다
git push	소스 코드의 변경 사항을 원격 저장소에 반영한다
git pull	원격 저장소의 변경 내용이 현재 디렉토리로 가져와진 뒤 병합된다
git merge	변경 사항 등이 확정 난 후 브랜치들을 병합한다. 작업 마무리 단계

6) git 설치

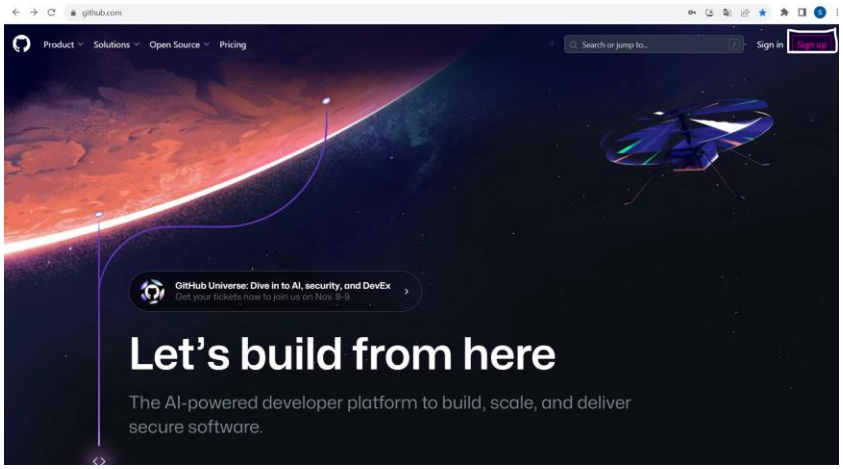
$ sudo apt-get update

$ sudo apt-get upgrade

$ sudo apt-get install git

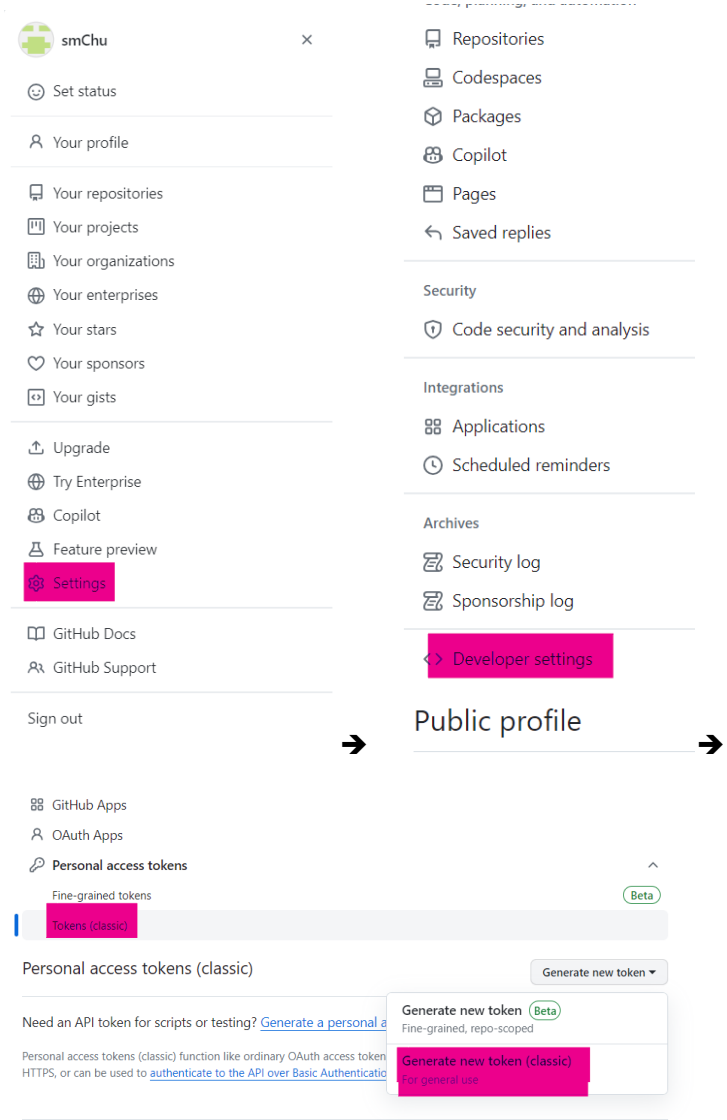
7) git 계정 생성

- https://github.com/ 에서 sign up 버튼 클릭

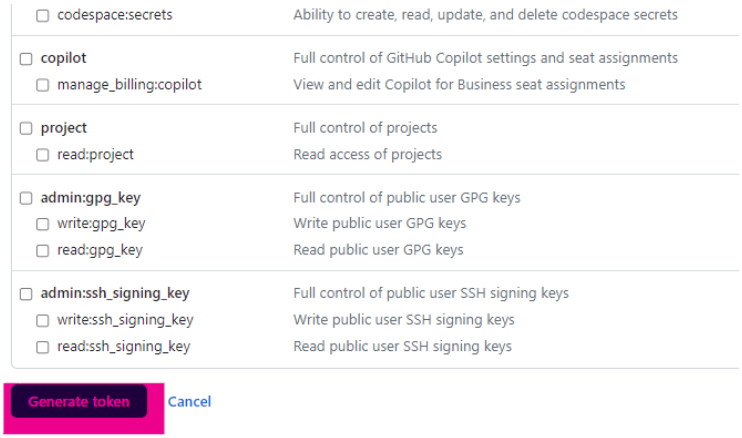


- 이메일, 비밀번호, 사용자 이름 입력

8) Git token 생성



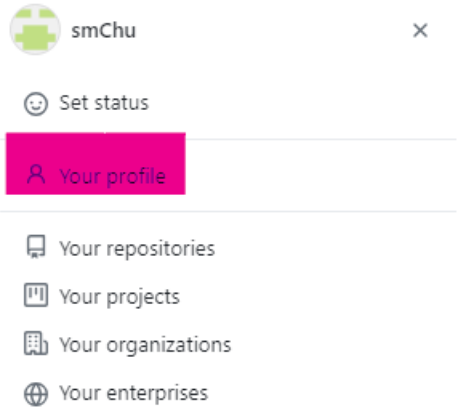
옵션을 선택 후 generate token 이라는 버튼을 눌러주면



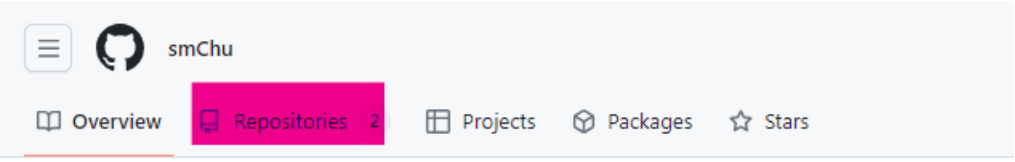
토큰 생성이 가능해진다. (personal access token)

9) repository 생성

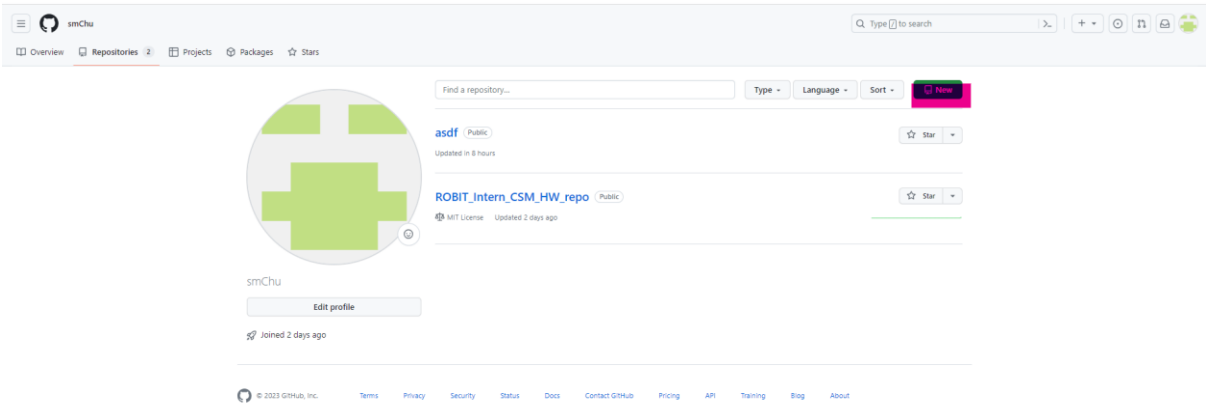
- 오른쪽 측면에 your profile 선택



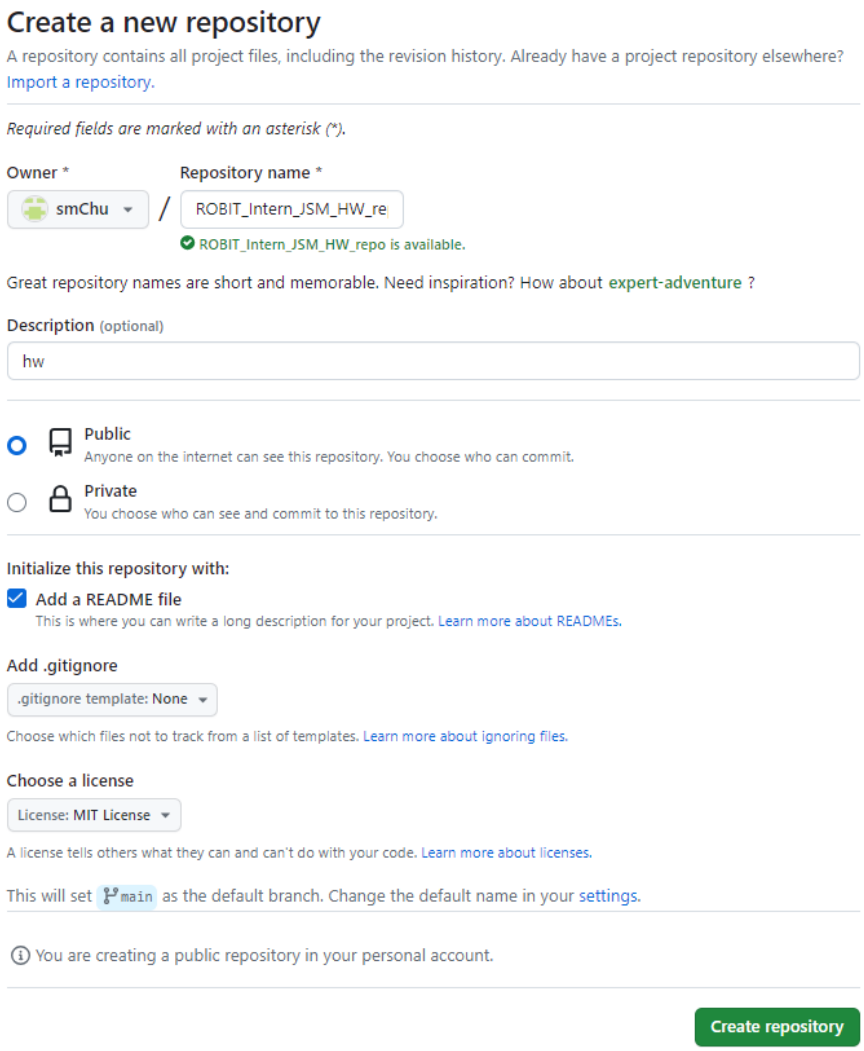
- 좌측 상단에 repository 선택



- 우측 상단의 new 버튼 선택



- 생성하고 싶은 레포지토리의 이름과 설명, 공개여부, README file 여부, 해당 라이선스를 선택 후 하단의 create repository 버튼 선택



10) github 로 push 하기

(1) 업로드 디렉토리를 정하고 그 경로에서 user name 과 user email 설정

[git config --global user.name "JSM"]

[git config --global user.email "chuandkim@gmail.com"]

(2) 저장소 생성

[git init]

```
chuseongmin@ChuSeongMin:~$ cd ROS
chuseongmin@ChuSeongMin:~/ROS$ git config --global user.name "JSM"
chuseongmin@ChuSeongMin:~/ROS$ git config --global user.email "chuandkim@gmail.com"
chuseongmin@ChuSeongMin:~/ROS$ git init
Reinitialized existing Git repository in /home/chuseongmin/ROS/.git/
```

(3) commit 할 repository 를 지정

[git remote add origin https://github.com/smChu/ROBIT_Intern_JSM_HW_repo]

[git fetch origin]

->정상적으로 연동

(4) git 의 현재 상태를 알고 push 할 수 있는 폴더 확인

[git status]

```
chuseongmin@ChuSeongMin:~/ROS$ git remote add origin https://github.com/smChu/ROBIT_Intern_CSM_HW_repo
fatal: remote origin already exists.
chuseongmin@ChuSeongMin:~/ROS$ git fetch origin
chuseongmin@ChuSeongMin:~/ROS$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        HW_003/
        TEST_FOLDER/

nothing added to commit but untracked files present (use "git add" to track)
```

(5) 커밋 메시지를 입력하고 업로드할 브랜치 입력

[git commit -m "커밋 메시지"]

[git push origin master]

```
chuseongmin@ChuSeongMin:~/ROS$ git add HW_003
chuseongmin@ChuSeongMin:~/ROS$ git commit -m "update"
[master 005f808] update
 11 files changed, 505 insertions(+)
 create mode 100644 HW_003/HW_003_001/image_viewer_qt/CMakeLists.txt
 create mode 100644 HW_003/HW_003_001/image_viewer_qt/include/image_viewer_qt/main_window.hpp
 create mode 100644 HW_003/HW_003_001/image_viewer_qt/include/image_viewer_qt/qnode.hpp
 create mode 100644 HW_003/HW_003_001/image_viewer_qt/launch/image_viewer_qt.launch
 create mode 100644 HW_003/HW_003_001/image_viewer_qt/package.xml
 create mode 100644 HW_003/HW_003_001/image_viewer_qt/resources/images.qrc
 create mode 100644 HW_003/HW_003_001/image_viewer_qt/resources/images/icon.png
 create mode 100644 HW_003/HW_003_001/image_viewer_qt/src/main.cpp
 create mode 100644 HW_003/HW_003_001/image_viewer_qt/src/main_window.cpp
 create mode 100644 HW_003/HW_003_001/image_viewer_qt/src/qnode.cpp
 create mode 100644 HW_003/HW_003_001/image_viewer_qt/ui/main_window.ui
chuseongmin@ChuSeongMin:~/ROS$ git add TEST_FOLDER
chuseongmin@ChuSeongMin:~/ROS$ git commit -m "update"
[master 9044f6c] update
 1 file changed, 1 insertion(+)
 create mode 100644 TEST_FOLDER/test
chuseongmin@ChuSeongMin:~/ROS$ git push origin master
Username for 'https://github.com': smChu
Password for 'https://smChu@github.com':
Enumerating objects: 28, done.
Counting objects: 100% (28/28), done.
Delta compression using up to 16 threads
Compressing objects: 100% (20/20), done.
Writing objects: 100% (27/27), 21.29 KiB | 3.55 MiB/s, done.
Total 27 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/smChu/ROBIT_Intern_CSM_HW_repo
   67a2bae..9044f6c  master -> master
```

(6) 오류 해결하기

[git pull origin master --allow-unrelated-histories]

(7)

[git push origin HEAD:master]

성공

```
chuseongmin@ChuSeongMin:~/ROS$ git push origin HEAD:master
Username for 'https://github.com': smChu
Password for 'https://smChu@github.com':
Everything up-to-date
```