



EDISON 플랫폼 소개

Summary: EDISON 플랫폼은 웹 브라우저 접속만으로 C, C++, Fortran, MPI, Python등으로 코딩된 Science Apps(시뮬레이션 SW)을 활용할 수 있는 환경을 제공합니다. 본 문서는 Science App 개발자들을 위한 개발 문서로 EDISON 플랫폼에서 시뮬레이션 SW를 개발하는 방법에 대해서 소개하고자 합니다.

EDISON 플랫폼 소개

EDISON 플랫폼은 웹 브라우저 접속만으로 C, C++, Fortran, MPI, Python등으로 코딩된 Science Apps(시뮬레이션 SW)을 활용할 수 있는 환경을 제공합니다. 본 문서는 Science App 개발자들을 위한 개발 문서로 EDISON 플랫폼에서 시뮬레이션 SW를 개발하는 방법에 대해서 소개하고자 합니다.

Science App 등록 절차

개발자 권한 신청 및 발급

Bulb 서버 접속 - Code upload & Compile - 실행 파일 압축

사이언스 앱스토어 > 앱 관리 > 앱 등록 - 앱 정보 입력 - 실행환경 정보 입력 - 입/출력 포트 정보 입력 - 앱테스트

서비스 요청 -> 관리자 승인 -> 서비스!

리눅스 기초

Summary:

리눅스 기초

EDISON 플랫폼은 리눅스 기반으로 개발되어 있어서 시뮬레이션 SW의 경우에도 리눅스에서 컴파일과 실행이 되는지 확인해야 합니다. 이를 위해 필요한 기본적인 내용들을 정리하였습니다.

리눅스 기본 명령어

명령어	설명	예제
ls	파일 리스트 보기	
cd	디렉토리를 변경	cd cgi-bin : 하부 디렉토리인 cgi-bin으로 들어감. cd .. : 상위 디렉토리로 이동 cd ~ : 어느곳에서든지 자기 홈디렉토리로 바로 이동
mkdir	디렉토리를 생성	mkdir download : download 디렉토리 생성
rm	파일 삭제	rm test.html : test.html 파일 삭제 rm -rf <디렉토리> : 디렉토리 전체를 삭제
pwd	현재 경로 보기	
chmod	퍼미션 조정	chmod 755 a.sh : a.sh파일의 퍼미션을 755로 조정
vi	vi 에디터 실행	vi newfile : vi 편집기 상태로 들어감

리눅스 필수 명령어

명령어	예제
./x	x 프로그램 실행 (현재 디렉토리에 있는 것)
↑/↓	이전에(↑) / 다음에(↓) 입력했던 명령어

명령어	예제
cd x	디렉토리 X로 가기
cd ..	한 디렉토리 위로 가기
x 다음 [tab] [tab]	x 로 시작하는 모든 명령어 보기
ls (또는 dir)	디렉토리 내부 보여주기
mv x y	파일 x를 파일 y로 바꾸거나 옮기기
cp x y	파일 x를 파일 y로 복사하기
rm x	파일 지우기
mkdir x	디렉토리 만들기
rmdir x	디렉토리 지우기
rm -r x	디렉토리 x를 지우고 하위도 다 지우기
exit	시스템 종료

- [레퍼런스](#)

Vim Editor

리눅스 시스템에 설치되어 있는 기본적인 워드 프로세서로 기본적인 사용법을 숙지하고 있어야 합니다. 본 가이드에서는 Vim Editor에 대한 세부적인 사용법에 대해서는 설명하진 않겠습니다. 아래 사이트를 참고하여, Vim Editor에 대한 사용법을 습득하길 바랍니다.

- [웹을 통해 vim 사용법을 배울 수 있는 튜토리얼 제공](#)
- [vim 관련 사용법에 대해 정리되어 있는 사이트\(한글\)](#)

리눅스에서 컴파일 하기

- [리눅스에서 C언어 컴파일 하기](#)

Makefile

리눅스 상에서 소스코드 컴파일을 편리하게 해주는 쉘 스크립트 파일입니다. 관련 상세한 정보는 아래 링크 사이트를 참고 하시길 바라며, 본 가이드에서는 makefile 샘플 파일에 대한 설명만 하도록 하겠습니다.

- makefile 구조 이해

Sample 예제 파일 구성

| 파일 명 | 설명 | - | - | main.c | main.c | | sub1.c, sub1.h | 라이브러리1 | | sub2.c, sub2.h | 라이브러리2 | | Makefile | Makefile file |

예제 파일

main.c

```
#include <stdio.h>
#include "sub1.h"
#include "sub2.h"

int main()
{
    myprint_sub1();
    myprint_sub2();
    return 1;
}
```

sub1.h

```
void myprint_sub1();
```

sub1.c

```
#include <stdio.h>
#include "sub1.h"
void myprint_sub1()
{
    printf("I am sub1\n");
}
```

sub2.h

```
void myprint_sub2();
```

sub2.c

```
#include <stdio.h>
#include "sub2.h"
void myprint_sub1()
{
    printf("I am sub2\n");
}
```

Makefile

```
# Makefile example
CC = gcc
FLAGS = -o

OBJS = main.o sub1.o sub2.o
SRCS = $(OBJS:.o=.c)

TARGET = test
all: $(TARGET)
$(TARGET) : $(OBJS)
    $(CC) $(FLAGS) $@ $(OBJS)
clean :
    rm -rf $(OBJS)
```

(1) `CC = gcc` 사용하고자 하는 컴파일러 명령어를 입력하면 된다. (icc, gcc, ifort, f90 등 ...)

(2) `FLAGS = -o` 컴파일 옵션을 넣어 주면 된다. 세부 옵션은 각 컴파일러 매뉴얼 참조 -o 옵션은 컴파일 결과 파일의 이름을 지정해주겠다는 명령어이다. 여기에서는 결과 파일명을 `TARGET =test`로 지정해주었다.

(3) `OBJS = main.o sub1.o sub2.o` 컴파일에 필요한 Object 파일과 Source 파일 명을 써주면 된다. 컴파일 과정에서 Object 파일은 자동으로 생성되며, (3)에 이름을 지정해 주어야 된다.

(4) `SRCS = $(OBJS:.o=.c)` 의 `$(OBJS:.o=.c)`는 `main.c sub1.c sub2.c`를 컴퓨터가 `OBJS`를 읽어 `.o`를 `.c`로 수정해 준 것이다.

fortran 의 경우에는 `$(OBJS:.o=.f90)`라고 입력하면 된다.

(5) `TARGET = test` 출력 파일 명 설정 CLASS에 `-o` 옵션을 지우면 안된다.

(6)

```
all: $(TARGET)
$(TARGET) : $(OBJS)
    $(CC) $(CLAGS) $@ $(OBJS)
```

make all 명령어를 설정하는 부분이며, 모든 Source 파일을 컴파일 하고 생성된 Object 파일들을 가지고 링크과정을 거쳐 실행 파일을 만드는 부분이다. `$(CC)` 앞에 `<tap>` 키를 입력해야 한다.

(7)

```
clean :
    rm -rf $(OBJS)
```

make clean 명령어를 설정하는 부분이며, 컴파일 과정에서 생성된 Object 파일을 지우는 부분.

- Makefile 실행하기
- 실행 파일 생성 : make all
- 생성된 Object 파일 삭제 : make clean

완성된 Makefile과 소스 파일을 가지고 실행 파일을 만들어 보면 아래와 같다.