펭귄 스터디 내용17

작성자 : 류성수
작성일자 : 24.07.05

url : https://www.youtube.com/watch?v=El5aeMs8Y-c&list=PLz--ENLG_8TMdMJIwyqDIpcEOysvNoonf&index=19

● Makefile

1. 이게 뭐지?

리눅스 및 유닉스 계열 운영체제에서 사용되는 빌드 자동화 도구인 make를 위한 구성 파일이다.
주로 소프트웨어 개발에서 프로젝트를 빌드(컴파일 및 링크)하는 작업을 자동화하는 데 사용된다.
Makefile은 프로젝트의 빌드 규칙과 종속성을 정의하는 일종의 스크립트이다.

2. 구성요소

요소	설명
target	빌드할 파일 또는 실행가능한 프로그램
dependencies	타겟을 빌드하기 위해 필요한 파일
commands	타겟을 빌드하기위해 실행되어야 하는 리눅스 명령어

3. 문법

makefile의 기본적인 문법은 다음과 같이 선언될 수 있다.

```
all : main.o sum.o sum2.o
	gcc -o test main.o sum.o sum2.o

main.o : main.c
	gcc -c main.c

sum.o : sum.c
	gcc -c sum.c

sum2.o : sum2.c
	gcc -c sum2.c
```

밑의 표에서 기본적으로 현재 디렉터리 내에 main.o, sum.o 가 있으면 실행될 수 있다. (종속성파일 존재)

<table>
<tr>
<td>

```
all : main.o sum.o
	gcc -o test_makefile main.o sum.o

main.o : main.c
	gcc -c main.c

sum.o : sum.c
	gcc -c sum.c
```

</td>
<td>

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy17# make
gcc -c main.c
gcc -c sum.c
gcc -o test_makefile main.o sum.o
```

</td>
</tr>
<tr>
<td>vim 에디터로 확인한 makefile</td>
<td>터미널창에서 실행 (sum2.c를 다시 뺐다)</td>
</tr>
</table>

“all :” 행을 살펴본다면,
all은 여기서 터미널 창에서 입력할 수 있는 터미널 명령어 (타겟) 이다.
그 뒤의 파일들은 all 명령을 실행시키기 위한 종속정 파일들이다.
밑의 빨간 글씨는 타겟을 빌드하기위해, 실행되어야 하는 리눅스 명령어이다.

4. 흐름

1. all (타겟)을 실행시키기 위해서, 종속성을 확인하는데, 혹시 밑에 종속성의 요소들이 타겟일 수도 있기에, 밑을 확인한다.
2. 현재 main.o 종속성파일이 밑에 타겟으로 설정되어 있기에, main.c 종속성 파일을 확인하는데 없어서, 디렉터리를 확인한다.
3. 현재 디렉터리에 종속성 소스파일이 있기에, main.o 타겟의 커맨드를 실행한다

(sum2.c 소스와 sum2.o 타겟을 없앴다)

4. sum.o 종속성파일이 밑에 타겟으로 있기에, main.o와같은 과정을 거친다.
5. 이제 all 타겟을 실행시킬 종속성파일이 완성되었기에, 해당 커맨드를 실행시킨다.

main.c

```c
#include <stdio.h>
#include "sum.h"

void main()
{
        printf("hello\n");
        printf("sum : %d\n",sum(1,2));
}
```

sum.c

```c
#include "sum.h"

int sum(int a, int b)
{
        return a+b;
}
```

이 두파일을 결합하여 컴파일 하여, .out 실행파일을 실행하면, 잘 나오는 것을 확인 할 수 있다.

```
Makefile  main.c  main.o  main.out  sum.c  sum.h  sum.o  test_makefile
root@LAPTOP-BQ4AK11N:/project/24-Cstudy17# ./test_makefile
hello
sum : 3
```

5. 변수

makefile에선 타겟의 명령어를 줄이기 위해 변수를 선언할 수 있다.

```
EXE = test_makefile.out
OBJS = main.o sum.o
```

실행파일과 필요한 종속성 파일을 전부 변수로 바꾸고 타겟의 커맨드를 다음과 같이 바꾸면 된다.

```
$(EXE) : $(OBJS)
        gcc -o test_makefile.out $^
# 이미 EXE와 필요한 파일들을 선언했기 때문에 명령어 사용가능
```

또한, 파일명만 변수로 설정하지 않고 컴파일러도 변수로 설정이 가능하다.

```
CC = gcc
++ = g++
```

그리고, 크로스 컴파일을 가정하여 , arm코어용 gcc를 쓰기도 한다면, 밑처럼도 선언 가능하다

```
# 크로스 컴파일의 경우
CROSS = arm-
CRCC = $(CROSS)gcc
# 그럼, CRCC를 사용시, arm-gcc 가 된다.
```

이렇게 위에 선언한 변수를 사용하려면, 변수를 소괄호로 감싸고 $를 앞에 붙이면 된다.

최종적으로 다음과 같이 간단하게 선언 할 수 있다.

```
$(EXE) : $(OBJS)
	$(CC) -o $@ $^
# gcc도 변수로 두고  $로 사용하였다.
```

6. 와일드 카드

프로젝트를 진행하면서 의존성 변수가 계속 늘어난다면, 변수에 요소를 추가해주고, 최종타겟을 실행하기위한 의존성 파일을 완성하는 타겟 구문도 계속 작성해 줘야 할 것이다.

이게 귀찮으니, 다음과 같이 선언할 수 있다.

```
C_SRC = $(wildcard *.c)
OBJS = $(C_SRC:.c = .o)
# 와일드카드 문법은 해당 디렉터리의 모든 소스파일들을 특정 변수로 추가시킨다는 뜻
# 밑의 문법은 소스파일들을 전부 obj 파일로 바꾼다는 뜻
# 앞으로 소스를 추가하면, 그냥 터미널에 추가하자마자 전부 다 컴파일되서 실행됨
```

이러면, 기본적으로 해당 디렉터리의 모든 C 소스파일들을 특정변수로 추가시키고
OBJ는 C소스를 obj 파일로 변환한다는 뜻이다.
이러면, 앞으로 프로젝트를 진행하면서, 비주얼 스튜디오처럼 F5 딸깍 한번으로 전부 실행시킬 수 있다.

7. 의존성 변수가 없는 타겟들은?

그냥 터미널에 make만 쳐서는 의존성 없는 타겟들은 실행되지 않는다.
예를들어서 다음과 같은 명령을 makefile에 추가하였다.

```
clean :
	rm *.o
# 의존파일이 없는 명령어들은 따로 make써주고 명령어를 적어주기
# make clean

delete :
	rm test_makefile

delete.out :
	rm *.out

good :
	echo good
```

이러한 타겟을 실행시키려면 다음과 같이 터미널에 입력하면된다.

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy17# make good
echo good
good
```