2009년 06월 28일

조금더~ Make 테크닉들

테크닉1) 이제까지는 한 Directory에 모든 object를 모아 놓아 make -C option을 이용할 필요가 없었는데, -C는 change directory를 의미합니다. make file을 여러 개 만들어 놓고, 상위 makefile에서 하위 make file들을 불러 제낄 때 -C option으로 Direcoty를 옮겨 가면서 make를 실행하도록 make를 구성할 수도 있답니다.

예를 들어볼까요?

```
DIRS = a b c
OBJECTS = a.o b.o c.o
TARGET = k

all : objs
	tcc -o $(TARGET) $(OBJECTS)

objs :
	@for dir in $(DIRS); do₩
	make -C $$ dir
	done
```

이렇게 짜는데, 이렇게 하면항상 directory를 등록해 줘야 하는 번거로움이 있죠. 명령어 앞에 @을 붙여주셨는데 이건 테크닉 5번에 나와요.

테크닉2) vpath
make에서 vpath를 지정해 주면,그것 보다 하위 directory의 c를 찾아서 자동으로 compile을 해줍니다. 이 말 어렵죠. vpath는 다음과 같은 형식으로 사용해요.

```
CSRC_PATH = c:/project/src
vpath %.c $(CSRC_PATH)
vpath %.h $(CSRC_PATH)
```

이렇게 선언해 주면, c나 h file을 만났을 때 vpath로 지정된 path (c:/project/src)에서부터 찾아간답니다. %는 dos의 *처럼 wild card 역할을 하고요. vpath는 다음에 다시 선언한다고 해서 나중에 선언된 걸로 replace되는 게 아니라, 계속 더해지는 개념이에요 차곡차곡. vpath에 지정된 path를 지우고 싶으면 그냥 vpath라고 명령을 주면 됩니다. vpath에 계속~ 넣어주기만 하면 된다니까요!

참고로, vpath는 이렇게 pattern을 지정해서 path를 정해 줄 수 있는데, pattern과 상관없이 현재 directory에 재료가 없을 때 search할 수 있는 directory를 정해 줄 수도 있는데, 그건 대문자 VPATH로 써줘요.

VPATH = c:/project/src:c:/project/src/obj 라고 정해 주고서,
executable.elf : recipe.o
라고 정해 준 다음 실행 했을 때, make가 있는 directory에 recipe.o가 없으면,
executable.elf : c:/project/src/executable.o 라고 해석해 주고 거기에도 없다면
executable.elf : c:/project/src/obj/executable.o 라고 해석해 주어, 어디에 있는 찾아 다니는 집요함을 보여줄 수 있어요.

테크닉3) include

source file만 include하느냐! make도 include할 수 있답니다. 하나의 mak file이 너무 길면, macro들만 모아서 file로 만든 후 본체 mak에서 include만 해주면 하나의 mak처럼 동작해요. 그리고, -include option이 있는데, 이 option으로 include하면 막상 include하려는 파일이 없어도 make가 error를 내지 않아요! 이 얼마나 편리한 시스템입니까.

테크닉4) Make file에는 C에서의 Macro 비스므리한 건 없느냐! 물론 있지요.

```
define 매크로
나불나불~
endef
```

요렇게 만들어 주고서 매크로 이름을 불러주면 됩니다. 저 같은 경우에는 Make를 debugging할 때 많이 쓰는데요. 예를 들면 변수 Macro를 확인할 때 자주 쓰는데요.

```
debugging :
        $(SHOW_THE_VARIABLES)

define SHOW_THE_VARIABLES
 @echo ----------------------------------------------------------------
 @echo my interestings
 @echo $(INCLUDE_PATHS) $(OBJS) $(CC)
 @echo ----------------------------------------------------------------
endef
```

뭐 이런 식으로 target중간중간에 끼워 넣고 그 값들을 확인한답니다. 역시나 printf style의 debugging이 최고로 좋다니깐요!

테크닉5) @ 명령 : 현재 line을 감춰주고 결과만 출력해 주어요.

```
all :
        @echo "It's me! make"
결과 :
   It's me! make
```

@ 기호가 없으면 출력은 다음과 같이 주지요. 현재 line하고, 결과하고요.

```
all :
        echo "It's me! make"
결과 :
 echo It's me! make
 It's me ! make
```

으흠~ 간단하지요?

테크닉 6) .PHONY → 가짜 타깃이라고 유명하죠. 흔히들 C code에서는 BOGUS라고 많이 쓰죠. 가짜라는 의미에요. 도대체 가짜 target이라는 건 뭔가요. 어흠. build의 대상이 아니라는 거에요. 즉, target이 아니라는 거죠. 그 말도 차암~ 이상하네요.

보통은 clean이라는 PHONY target을 target에 많이 쓰는데요. .PHONY : clean 이라고 선언하고서,
clean : rm *.o 라고 target을 정의해 주었을 때 make clean 명령을 주면 clean은 rm *.o가 clean을 하기 위한 재료이긴 하지만, clean이라는 target file을 만들어 내지는 않는 다는 거지요. 어쨌거나. 이제부터 clean은 build의 대상이 아니기 때문에, 그 재료 중에 clean이라는 게 있어도, clean 파일을 만들어 내지 않아요. 이런 식으로 .PHONY target을 이용해서 명령어들을 주르륵 실행시키고 싶을 때 많이 이용한답니다. 그러니까, make clean 이라는 명령을 내렸을 때 clean이라는 file이 있는지 없는지는 make가 전혀 신경 쓰지 않고, 무작정 그 뒤에 있는 명령어들을 실행할 수 있는 거에요. 대신 주의할 것은 .PHONY target은 다른 target의 재료가 되어서는 안 된다는 rule이 있어요. 주의해 주세요.

테크닉 7) Flags : make file에서도 c file에서와 같이 #ifdef 같은 류의 선택적 실행이 가능합니다.
C에서는

#ifdef WINDOWS
윈도우 코드 나불
#endif
뭐 이런 식으로 WINDOWS가 #define으로 선언되어 있는 경우 윈도우 코드 나불을 선택적 컴파일 할 수 있지요. 이것과 마찬가지로 make에서도 #ifdef 을 쓸 수가 있어요.

#ifdef WINDOWS
윈도우 make 나불
endif

이런 식으로 make를 만들어 놓고, WINDOWS=yes 를 위에 선언하던가, command 라인에서 make 명령을 줄때, make WINDOWS=yes 이런 식으로 명령을 주면 선택적 실행을 할 수 있다니까요 .

비스므리한게 이거 말고 또 재미있는 명령어가 있는데요, 그건 Macro의 값을 비교할 수 있는 명령어에요. ifeq ~ endif 라는 건데요. 어떻게 쓰는 거냐면,

ifeq ($(WINDOWS),yes)
윈도우 make 나불~
endif
이런 형식으로 사용하고요, 이런 경우에는 WINDOWS가 선언되어 있는 게 문제가 아니고, yes이어야 중간에 명령어를 실행할 수 있습니다. 반대로는 not equal도 쓸 수 있는데, ifneq ($(WINDOWS), yes) ~ endif
로 쓰게 되면 WINDOWS가 yes가 아닌 경우에만 쓰게 되고요. 위의 선택적 실행은 당돌빠따로 else도 쓸 수 있답니다.

테크닉8) 이제 마지막으로 Text를 조작할 수 있는 make에는 많이 준비되어 있지요. 좌르르르륵 예들을 들어보죠 머.

$(subst a,b,text) : text에서 a를 b로 교체할 수 있어요.
$(patsubst pattern,rep,text) : text에서 pattern을 찾아서 rep로 replace해요.
$(strip string) : string안에서 공백문자들을 제거해요.
$(addsuffix suffix,name) : name에다가 suffix를 뒤에 더 붙여 줘요.
$(addprefix prefix,name) : name 앞에다가 prefix를 붙여주지요.
$(findstring find,text) : text에서 find라는 string을 찾아요.
$(dir name) : name에서 directory부분을 빼내주어요.
$(notdir name) : name에서 directory가 아닌 부분을 빼내주어요. 보통 file name 뽑아낼 때 쓰죠.
$(suffix name) : name에서 확장자를 추출해 내요.
$(basename name) : name에서 확장자를 뺀 file이름을 빼내주지요.
$(word n,text) : text중에서 n 번째 단어를 추출해 주어요.
$(words text) : text가 몇 개 단어로 이루어져 있는지 봐주지요.
$(wildcard pattern) : 지정된 Directory에서 pattern과 일치하는 file들을 가져오지요. 디렉토리를 지정하지 않으면 현재 dir이에요.
$(shell command) : Shell 명령어를 실행해 주고 그 결과를 가져오지요.
$(origin variable) : Macro가 어떻게 정의되었는지를 알려줘요. 이걸로 debugging을 하기도 해요.
$(foreach var,words...,text) : var에 각각의 word를 넣고서 text에 적용하는 거에요. 말이 좀 어렵지만, 보통 text에는 $(var)에 관련된 내용이 같이 들어 있기 마련이죠. 어렵다.
몇 개 예만 들어 볼께요.
보통 addprefix 같은 경우는 어디에 쓰이는가 보겠사옵니다.
'
OBJS := $(addprefix $(TARGETDIR)/, $(SOURCES:%.c= %.o))

이렇게 하면 SOURCES로 등록된 모든 c file의 확장자를 ,o로 바꾸고, 그 앞에다가는 특정 directory를 붙여 주겠죠. 그러면 일관된 directory에 ,o file들을 모아놓을 수 있답니다.

또, foreach를 해볼까요.

SOURCES = $(foreach dir, recipes, spaghetti, ramen, $(wildcard $(dir)/*.c))
라고 해보시죠.그러면 어떻게 될까요?
결과는

recipes/recipes.c spaghetti/spaghetti.c ramen/ramen.c
요렇게 된답니다.

그러니까 각각의 recipes, spaghetti, ramen을 dir에 넣고서 맨 뒤의 text에 적용해서 결과를 가져옵니다요. 이해가 되려 나요. 쩝

테크닉 9) 다중 target → Multiple target
Make에서 target을 정할 때 한번 만 정해야 하느냐? 그것도 아닙니다. 결과를 하나만 만들어 내라고 한다면, 좀 심심하겠죠. target을 여러 번 넣을 수도 있어요. 예를 들면, target중에 setup이라는 target을 정하는데, setup : firmware dsp 라고 한번 선언한 후에도 뒤에 setup : dsp mcu
뭐 이런 식으로 또 선언할 수 있어요. 엮기고 엮인 관계죠.
fyi) firmware dsp mcu 모두 target이에요.
target의 재료로 target이 들어갈 수 있다는 말씀이시죠.

어쨌거나 make file에 대한 고급내용은 reference site가 있어요. **http://www.viper.pe.kr/docs/make-ko/make-ko_toc.html** 요기 가서 보시면 길 다라~안 contents가 보여요.더 자세한 내용들이 있으니까 잘 살펴 보시면 복 받으시겠죠 뭐.

그래도 이제까지 살펴본 make 내용만 잘 보셔도, 웬만한 make 작성뿐 아니라 해석도 대충 가능하시리라 믿고, 꼴 때리는 make는 이만 마무리에요~

directory, option, file, 실행, make, makefile, arm, compile, CrossCompile

Linked at at 2009/06/28 15:35

... e은 뭘하는 녀석일까~ ⓡ 컴파일을 더더더 쉽게. MACRO와 SUFFIX ⓢ 조금 더 Make 테크닉들 ⓣ Make option들 4) ARM 미장센 - ARM 제어의 구현   ... more