MAKE - 프로젝트 빌드

안양대학교 컴퓨터공학과 하은용

GNU make utility

내용

- make utility
- Makefile

make

make 도구

- make 도구의 핵심은 대규모 프로그램의 구성 모듈들 중 어느 것이 다시 컴파일 되어야 하는지 자동으로 결정하여 그들을 컴파일
  - 프로그램의 컴파일 시간을 줄이고 관리가 용이해지며, 단순 반복 작업을 최소화
- 기본적으로 자동으로 프로그램 코드를 컴파일 하기 위한 도구이므로 컴파일 명령의 실행이 주된 동작이지만, 컴파일이 아닌 단순 쉘 명령까지도 수행 가능

make의 동작 방식

- Makefile이라는 이름의 파일에 수행조건과 명령을 기술하면 make가 파일에 기술된 대로 명령들을 수행
  - Makefile, makefile, GNUmakefile 중 먼저 발견되는 파일에 기술된 대로 수행
  - 만약 다른 이름으로 기술되어 있다면 "make -f 파일명"으로 실행
- Makefile에 기술된 명령과 명령 수행조건이란
  - 명령 수행조건이 되는 파일 존재 유무, 최근 갱신 시간을 파악하여 명령을 수행

의존성과 Makefile (1)

diary.h

```
int memo();
int calendar();
```

memo.c

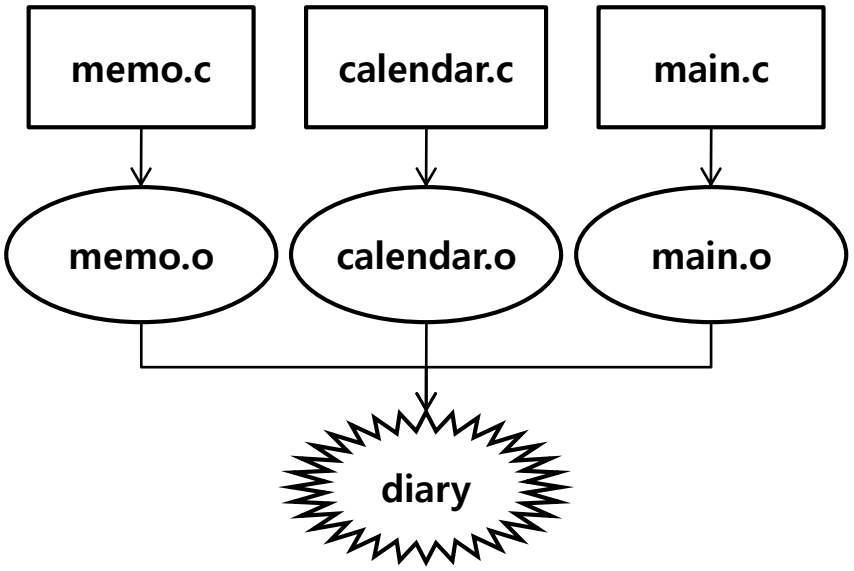
```
#include <stdio.h>
int memo() {
    printf("func memo.\n");
    return 0;
}
```

calendar.c

```
#include <stdio.h>
int calendar() {
    printf("func calendar.\n");
    return 0;
}
```

main.c

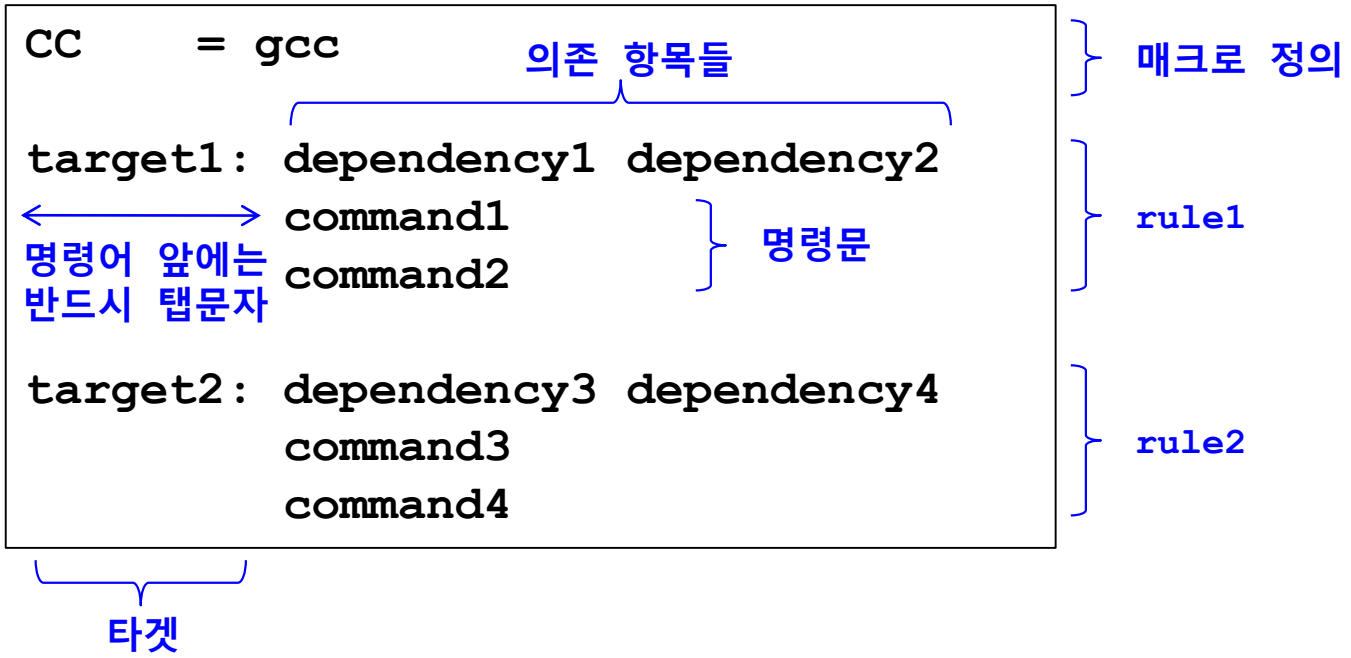
```
#include "diary.h"
int main() {
    memo();
    calendar();
    return 0;
}
```



Makefile의 기본 구조

동작 규칙

- Target file이 존재하지 않거나, target file보다 의존 항목 file이 더 최신일 경우 명령 수행
- 명령이 수행되기 직전에 의존성 항목을 target으로 하는 규칙 검사



Makefile 작성 규칙

- 룰의 명령은 탭문자로 시작
- 룰의 명령문들 사이의 공백은 무시
- '#'은 line comment
- 행의 마지막 '\' 문자는 다음 행과 연속 행을 의미
- 종속 항목이 없는 target 가능
- 명령 부분에는 어떠한 명령어도 가능

의존성과 Makefile (2)

Makefile 작성

```
all: diary

diary: memo.o calendar.o main.o
	gcc -o diary memo.o calendar.o main.o

memo.o: memo.c
	gcc -Wall -Wextra -c -o memo.o memo.c

calendar.o: calendar.c
	gcc -Wall -Wextra -c -o calendar.o calendar.c

main.o: main.c
	gcc -Wall -Wextra -c -o main.o main.c

clean:
	rm -rf *.o diary
```

- 제일 처음 target 발견, all을 만들기 위해 종속항목 diary 존재 유무 확인
- diary를 만들기 위해 종속항목들의 존재 유무 확인
- 탭
- target과 종송항목간의 조건에 따라 실행
- memo.o, memo.c 존재 유무, 생성 시간 비교, memo.c의 존재유무와 생성시간은 target으로서 연쇄적으로 다시 확인
- 이 target은 처음 target all로부터 연쇄적으로 연결되지 않으므로 make 명령시 명시적으로 target을 지정해야만 처리됨. 또한 의존 항목이 없으므로 무조건 명령 실행

의존성과 Makefile (3)

컴파일

```
$ ls
Makefile  calendar.c  diary.h  main.c  memo.c
$ make
gcc -Wall -Wextra -c -o memo.o memo.c
gcc -Wall -Wextra -c -o calendar.o calendar.c
gcc -Wall -Wextra -c -o main.o main.c
gcc -o diary main.o memo.o calendar.o
$ ls -F
Makefile  calendar.c  calendar.o  diary*  diary.h  main.c  main.o
memo.c  memo.o
$ ./diary
func memo
func calendar
$ make
make: `all'를 위해 할 일이 없습니다.
$ make clean
rm -f *.o diary
$
```

의존성과 Makefile (4)


```
$ ls
Makefile  calendar.c  diary.h  main.c  memo.c
$ make
gcc -Wall -Wextra -c -o memo.o memo.c
gcc -Wall -Wextra -c -o calendar.o calendar.c
gcc -Wall -Wextra -c -o main.o main.c
gcc -o diary main.o memo.o calendar.o
$ touch memo.c
$ make
gcc -Wall -Wextra -c -o memo.o memo.c
gcc -o diary main.o memo.o calendar.o
$ touch main.c
$ make
gcc -Wall -Wextra -c -o main.o main.c
gcc -o diary main.o memo.o calendar.o
$ make clean
rm -f *.o diary
$ make memo.o
gcc -Wall -Wextra -c -o memo.o memo.c
$
```

매크로 정의 규칙 (1)

- 매크로란 '='를 포함하는 하나의 문장

```
NAME  = string
```

- '#'은 한 줄 주석문의 시작

```
NAME  = string   # 주석
```

- '\' 여러 행을 하나로

```
NAME = -D__KERNEL__ -DMODULE -Wall -Wextra \
       -DNO_DEBUG
```

- 매크로 참조는 소괄호나 중괄호로 둘러싸고 앞에 '$'

```
NAME = string
${NAME}         # string
$(NAME)         # string
${NAME}.c       # string.c
macro_$(NAME) # macro_string
```

매크로 정의 규칙 (2)

- 정의되지 않은 매크로 참조를 null값으로 처리

```
NAME   = string
MY_$(UNAME)      # MY_
```

- 중복 정의된 매크로는 마지막 정의된 값

```
NAME   = stringA
NAME   = stringB
$(NAME)             # stringB
```

- 매크로 정의시 매크로 참조 가능

```
NAME   = string
NAME2 = my $(NAME)   # NAME2는 my string으로 정의됨
```

매크로 정의 규칙 (3)

- 대입 규칙:
  - 재귀적 확장 매크로 (=) : 전방/후방 참조
  - 단순 확장 매크로 (:=) : 전방 참조

```
A    = $(B) BB
B    = $(C) CC
C    = D

a   := $(b) bb
b   := $(c) cc
c   := d

all:
     @echo $(A)
     @echo $(a)
```

```
$ make
D CC BB
bb
```

매크로 정의 규칙 (4)

대입규칙: +=

- 이전 정의 내용 다음에 공백을 하나 추가하여 덧붙임

```
NAME2 := string1
NAME2 += string2   # string1 string2
```

대입규칙: ?=

- 현재 정의하는 매크로가 정의되지 않았을 때 정의

```
NAME1 = my
NAME1 ?= string1   # NAME1은 my
NAME2 ?= string2   # NAME2는 string2
```

매크로 사용시 주의 사항

- 따옴표는 문자열의 일부가 아님

```
NAME  = "string"
# NAME은 "string" 아니라 string
all:
    @echo $(NAME)
```

- 매크로 이름에 ':', '=', '#' 사용 불가, 탭시작 불가

```
    NAME  = string        # X
NAME:2    = string        # X
NAME#2    = string        # X
```

매크로를 사용한 Makefile


```
CC      = gcc
CFLAGS = -Wall -Wextra
TARGET = diary

all: $(TARGET)

$(TARGET): memo.o calendar.o main.o
	$(CC) -o $(TARGET) memo.o calendar.o main.o

memo.o: memo.c
	$(CC) $(CFLAGS) -c -o memo.o memo.c

calendar.o: calendar.c
	$(CC) $(CFLAGS) -c -o calendar.o calendar.c

main.o: main.c
	$(CC) $(CFLAGS) -c -o main.o main.c

clean:
	rm -rf *.o $(TARGET)
```

내장 매크로

내장 매크로

- make 내부에 미리 정의된 매크로로 별도 정의 없이 사용 가능
- 내장 매크로 목록 확인 명령

```
$ make -p | grep ^[[:alpha:]]*[[:space:]]*=[[:space:]]
```

매크로	설명	기본값	매크로	설명	기본값
AR	아카이브관리도구	ar	ARFLAGS	ar 플래그	rv
AS	어셈블러	as	ASFLAGS	as 플래그	
CC	C컴파일러	cc	CFLAGS	C컴파일러플래그	
CXX	C++컴파일러	g++	CXXFLAGS	C++컴파일러플래그	
CPP	C전처리기	cc –E	CPPFLAGS	C전처리기플래그	
LD	링커	ld	LDFLAGS	링커플래그	
LEX	스캐너생성기	lex	LFLAGS	Lex플래그	
YACC	파서생성기	yacc	YFLAGS	Yacc플래그	
TEX	TeX문서 변환기	tex	RM	파일 삭제 명령	rm –f

자동 매크로 (1)

자동 매크로란

- 내장 매크로와 마찬가지로 make 내부적으로 정의되어 있으나, make –p 명령으로 확인할 수 없음
- Makefile 내부에서 부분적 내용에 따라 자동으로 변경됨

매크로	설명	예) /temp/target: /home/mylinux/test.c
$?	현재 타겟보다 최근에 변경된 종속항목 목록 (확장자 규칙에서 사용 불가)	$? → /home/mylinux/test.c
$^	현재 타겟에서 종속항목 목록 (확장자 규칙에서 사용 불가)	$(^F) → test.c
$@	현재 타겟의 이름	$(@F) → target
$<	현재 타겟보다 최근에 변경된 종속항목 목록 (확장자 규칙에서만 사용 가능)	$(<D) → /home/mylinux
$*	현재 타겟보다 최근에 변경된 현재 종속항목 이름, 확장자 제외 (확장자 규칙에서만 사용 가능)	$(*F) → test

자동 매크로 (2)

예제1

자동 매크로를 사용하지 않은 룰

```
target: dependency1.c dependency2.c
        gcc -o target dependency1.c dependency2.c
```

자동 매크로를 사용한 룰

```
target: dependency1.c dependency2.c
        gcc -o $@ $^
```

내장 매크로와 자동 매크로를 사용한 룰

```
target: dependency1.c dependency2.c
        $(CC) -o $@ $^
```

자동 매크로 (3)

매크로와 자동 매크로를 사용한 예제

```
CC      = gcc
CFLAGS = -Wall -Wextra
TARGET = diary

all: $(TARGET)

$(TARGET): memo.o calendar.o main.o
	$(CC) -o $@ $^

memo.o: memo.c
	$(CC) $(CFLAGS) -c -o $@ $^

calendar.o: calendar.c
	$(CC) $(CFLAGS) -c -o $@ $^

main.o: main.c
	$(CC) $(CFLAGS) -c -o $@ $^

clean:
	$(RM) *.o $(TARGET)
```

확장자 규칙 (1)

내장된 확장자 규칙

- 확장자에 따라 컴파일을 위한 컴파일러와 컴파일 규칙이 미리 정해져 있음
  - 확인 명령: $ make -p

```
%.o: %.f
        $(COMPILE.F) $(OUTPUT_OPTION) $<

%.o: %.c
        $(COMPILE.c) $(OUTPUT_OPTION) $<

%.o: %.cc
        $(COMPILE.cc) $(OUTPUT_OPTION) $<

COMPILE.c = $(CC) $(CFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c
OUTPUT_OPTION = -o $@
```

확장자 규칙 (2)

확장자 규칙을 사용한 예제

```
OBJECTS = memo.o calendar.o main.o
CFLAGS  = -Wall -Wextra
TARGET  = diary

all: $(TARGET)

$(TARGET): $(OBJECTS)
	$(CC) -o $@ $^

clean:
	$(RM) *.o $(TARGET)
```

```
$ make
cc -Wall -Wextra   -c -o memo.o memo.c
cc -Wall -Wextra   -c -o calendar.o calendar.c
cc -Wall -Wextra   -c -o main.o main.c
cc -o diary memo.o calendar.o main.o
$
```

확장자 규칙 (3)

확장자 규칙 재정의

- 확장자에 따른 규칙을 변경하고자 할 경우
  - 다음 예에서와 같이 단순히 컴파일 옵션만을 변경할 경우는 CFLAGS를 변경하는 것이 바람직. 이 예는 규칙 재정의를 설명하기 위한 것 뿐.

```
OBJECTS = memo.o calendar.o main.o
CFLAGS  = -Wall -Wextra
TARGET  = diary

.SUFFIXES: .o .c
%.o: %.c
	$(CC) $(CFLAGS) -DDEBUG -c -o $@ $<

all: $(TARGET)

$(TARGET): $(OBJECTS)
	$(CC) -o $@ $^

clean:
	$(RM) *.o $(TARGET)
```

.SUFFIXES는 특수내장타겟

매크로 대치

매크로 대치를 통한 다량의 파일 지정

- 컴파일해야할 파일이 많은 경우 일일이 목적파일명을 기술하는 것이 매우 힘듬

```
SRCS     = $(wildcard *.c)
OBJECTS = $(SRCS:.c=.o)
CFLAGS   = -Wall -Wextra
TARGET   = diary

all: $(TARGET)

$(TARGET): $(OBJECTS)
	$(CC) -o $@ $^

clean:
	$(RM) *.o $(TARGET)
```

"$(wildcard *.c)"는 현재 디렉터리에서 ".c"로 끝나는 모든 파일을 찾아 공백을 구분문자로 SRCS 매크로 정의
"SRCS = memo.c calendar.c main.c"와 동일

"$(SRCS:.c=.o)"는 SRCS 값에서 ".c"라는 문자는 ".o"로 대입된 값으로 결국 OBJECTS =memo.o calendar.o main.o"와 동일

더미 타겟

더미 타겟(Dummy target)의 사용

- 더미 타겟은 의존 항목들이 없고 명령 실행에 의해 타겟이 생성되지 않는 룰
- 타겟이 생성되지 않으므로 target지정에 의한 make 명령으로 룰의 명령은 매번 항상 실행

```
SRCS     = $(wildcard *.c)
OBJECTS = $(SRCS:.c=.o)
CFLAGS   = -Wall -Wextra
TARGET   = diary

all: $(TARGET)

$(TARGET): $(OBJECTS)
	$(CC) -o $@ $^

clean:
	$(RM) *.o $(TARGET)
```

```
$ make clean
rm -f *.o diary
$
```

명령어 사용 규칙

- 룰의 명령은 독립 쉘에서 실행
  - 따라서 다음과 같은 make 룰의 사용은 주의

```
del:
        cd ./backup
        rm -rf *
```

    - 현재 디렉터리 내의 모든 파일과 디렉터리가 삭제되는 결과를 초래
    - 다음과 같이 해야 원래 의도대로 backup 디렉터리 내의 파일 삭제

```
del:
        cd ./backup; rm -rf *
```

    - backup 디렉터리가 없는 경우 여전히 문제 발생

```
del:
        cd ./backup && rm -rf *
```

요약

- make를 사용한 컴파일 과정
  - Makefile 작성
  - make 명령 실행
    - make
    - make 타겟
    - make –f makefile명
- 매크로
  - 매크로란 '='를 포함하는 하나의 문장
  - '₩'은 두 행을 하나로 이어줌
  - 매크로 참조는 $(NAME)
  - 정의되지 않은 매크로 참조 값은 null
  - 중복 정의 매크로는 마지막 값
  - 매크로 정의시 매크로 참조 가능
- 매크로 대입 규칙
  - 재귀적 확장 매크로/단순 확장 매크로
  - +=, ?=
- 내장 매크로
  - AR, AS, CC, CXX, CPP, LD
  - ARFLAGS, CFLAGS, CPPFLAGS, LDFLAGS, RM
- 자동 매크로
  - $?, $^, $@, $<
- 매크로 대치
  - $(wildcard *.c), $(SRCS:.c=.o)
- 기타
  - 더미타겟, 명령어 사용 규칙