



Make 유틸리티

로봇SW 교육원

최상훈(shchoi82@gmail.com)

학습 목표

2

- make 유틸리티 이해
- Makefile 작성

make

3

- make
 - 프로젝트 관리 유틸리티
 - 컴파일 시간 단축
 - 파일의 종속 구조를 빠르게 파악할 수 있음
 - 기술 파일(Makefile)에 기술된 대로 컴파일 명령 또는 셸(shell)명령을 순차적으로 수행함
- make 를 사용하지 않을 경우
 - main.c, choi.c, kim.c lee.c 4개의 소스 파일로 구성된 프로그램일 경우

```
$ gcc -Wall -W -c lee.c
$ gcc -Wall -W -c kim.c
$ gcc -Wall -W -c choi.c
$ gcc -Wall -W -c main.c
$ gcc -Wall -W lee.o kim.o choi.o main.o -o prog
```

기술 파일(Makefile)의 구조

4

- 기술 파일(Makefile)의 구성

```
macro

target : dependency1 dependency2 ...
|.....| command1
|.....| command2
      ⋮
```

Tab으로 시작

- 매크로 정의 부분
 - 자주 사용되는 문자열 정의
- 규칙(Rule) 정의 부분
 - 타겟 (target) : 생성할 파일
 - 종속항목 (dependency) : 타겟을 만들기 위해 필요한 파일
 - 명령 (command) : 타겟을 만들기 위해 필요한 명령
 - ※ 타겟과 종속항목은 보통 파일명

make 동작과정

5

- make 명령 실행 시 make는 현재 디렉토리에서(CWD) Makefile 파일을 찾음
- make 실행 시 타겟을 지정할 수 있음
 - 지정하지 않으면 제일 처음 타겟을 수행함

make 동작과정

6

- 종속 항목이 없을 경우
 - 타겟 파일이 존재하는지 확인 후 없을 경우에만 명령 (COMMAND)을 실행
- 종속 항목이 있을 경우
 - 종속 항목을 순서대로 점검
 - 종속 항목 파일이 존재하지 않는 경우
 - 종속 항목 파일을 만들기 위해 규칙을 찾아 실행, 규칙이 없으면 오류(종료)
 - 종속 항목 파일이 존재하는지 경우
 - 규칙이 있는지 확인, 있으면 규칙 실행, 없으면 정상으로 간주하고 다음 종속 항목 점검
 - 모든 종속 항목을 점검한 후 타겟과 마지막 수정시간(들)을 비교
 - 종속 항목 파일이 더 최신일 경우 타겟을 만들기 위해 명령(COMMAND)을 실행
 - 최신여부의 판단
 - » 파일의 마지막 수정시간(파일의 stat 구조체 st_mtime를 확인) 비교
 - » 더미타겟일 경우는 항상 최신
 - 타겟 파일이 없을 경우 명령(COMMAND)을 실행
- 명령(COMMAND) 실행
 - 명령 (COMMAND) 실행이 모두 정상이면 타겟이 정상적으로 생성 되었다고 간주함
 - 명령 (COMMAND) 실행 중 오류가 발생되면 타겟이 정상적으로 생성되지 않았다고 간주함
 - 종속항목 점검 중 종속항목을 만드는 규칙에서 명령(COMMAND)실행 중 실패 시 중단됨(종료)

기술 파일(Makefile)의 작성

7

- Makefile 작성
 - 명령의 시작은 반드시 Tab으로 시작함
 - 비어있는 행은 무시됨
 - # 으로 시작 : 한줄 주석
 - \ 다음 줄로 이을 수 있음
 - ; 으로 명령라인을 나눌 수 있음
 - 종속 항목이 없는 타겟도 사용 가능
 - 명령 부분에는 어떤 명령이 와도 상관없음

실습 1-1: make

8

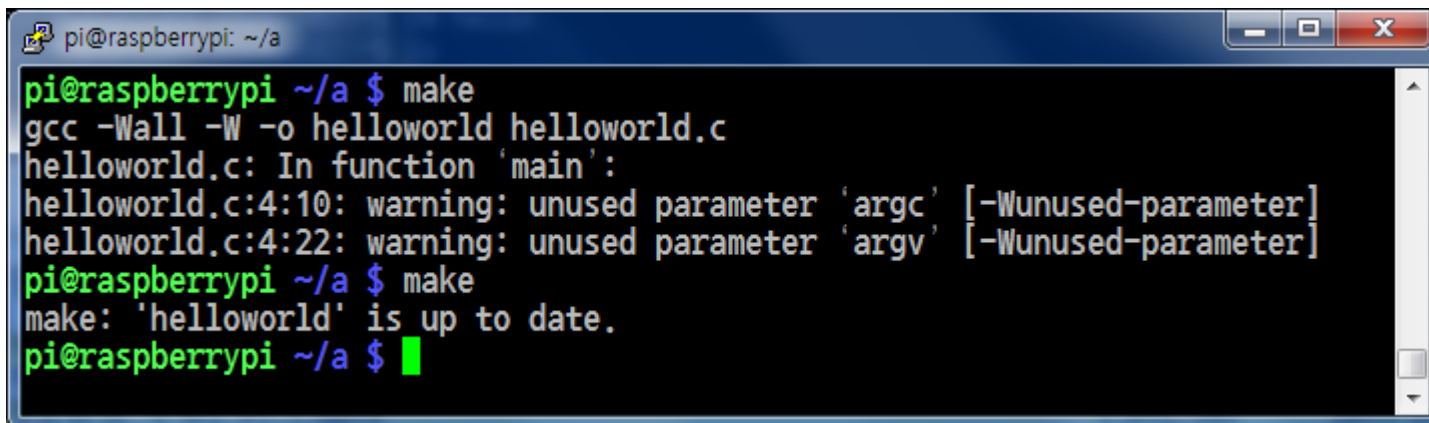
- Makefile 작성

```
$ vim Makefile
```

```
helloworld :  
    gcc -Wall -W -o helloworld helloworld.c
```

- make 실행

```
$ make
```



```
pi@raspberrypi: ~/a $ make  
gcc -Wall -W -o helloworld helloworld.c  
helloworld.c: In function 'main':  
helloworld.c:4:10: warning: unused parameter 'argc' [-Wunused-parameter]  
helloworld.c:4:22: warning: unused parameter 'argv' [-Wunused-parameter]  
pi@raspberrypi ~/a $ make  
make: 'helloworld' is up to date.  
pi@raspberrypi ~/a $
```


실습 1-2: make

9

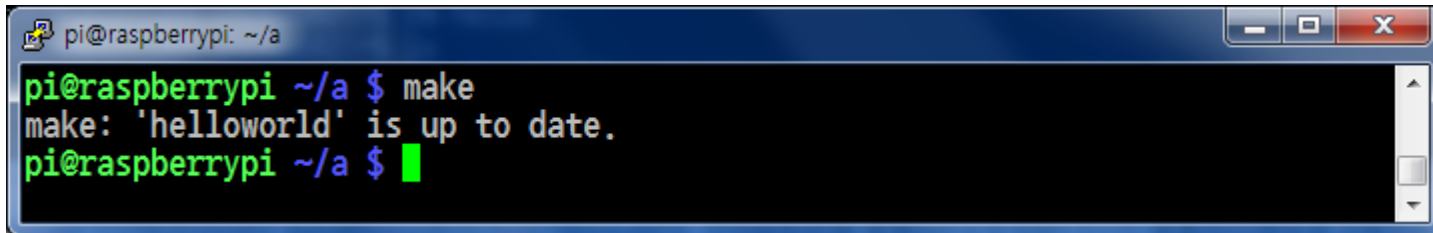
- helloworld.c 파일 수정

```
$ vim helloworld.c
```

```
printf("hello world\n"); -> printf("hello world2\n");
```

- make 실행

```
$ make
```

A terminal window titled 'pi@raspberrypi: ~/a' with standard window controls. The terminal shows the command 'make' being executed. The output is 'make: 'helloworld' is up to date.' followed by a new prompt line.

```
pi@raspberrypi ~/a $ make
make: 'helloworld' is up to date.
pi@raspberrypi ~/a $
```

실습 1-3: make

10

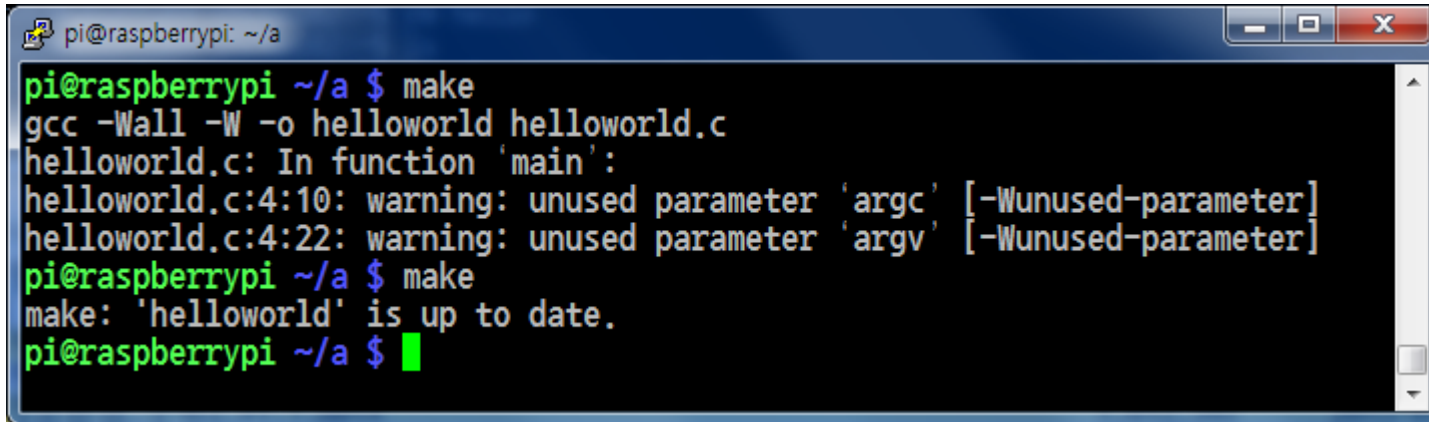
- Makefile 수정

```
$ vim Makefile
```

```
helloworld : helloworld.c  
    gcc -Wall -W -o helloworld helloworld.c
```

- make 실행

```
$ make
```



```
pi@raspberrypi: ~/a $ make  
gcc -Wall -W -o helloworld helloworld.c  
helloworld.c: In function 'main':  
helloworld.c:4:10: warning: unused parameter 'argc' [-Wunused-parameter]  
helloworld.c:4:22: warning: unused parameter 'argv' [-Wunused-parameter]  
pi@raspberrypi ~/a $ make  
make: 'helloworld' is up to date.  
pi@raspberrypi ~/a $
```

실습 1-4: make

11

- touch 명령으로 파일의 시간을 현재시간으로 업데이트

```
$ touch helloworld.c
```



```
pi@raspberrypi: ~/a
pi@raspberrypi ~/a $ touch helloworld.c
pi@raspberrypi ~/a $ make
gcc -Wall -W -o helloworld helloworld.c
helloworld.c: In function 'main':
helloworld.c:4:10: warning: unused parameter 'argc' [-Wunused-parameter]
helloworld.c:4:22: warning: unused parameter 'argv' [-Wunused-parameter]
pi@raspberrypi ~/a $ make
make: 'helloworld' is up to date.
pi@raspberrypi ~/a $
```

더미 타겟

12

- Dummy target 또는 Phony target
- **파일이 생성되지 않는 개념적인 타겟**

```
helloworld : helloworld.c
             gcc -Wall -W -o helloworld helloworld.c

clean :
        rm helloworld

hi :
        echo hello~

ping :
        echo pong

list :
        ls -al *.c

backup :
        ls helloworld.c
        cp helloworld.c helloworld_bak.c

everything : hi ping list backup
```

실습 2: make

13

- Makefile 수정


```
$ vim Makefile
```

```
helloworld : helloworld.c
    gcc -Wall -W -o helloworld helloworld.c

clean :
    rm helloworld
```

- clean 타겟 실행

```
$ make clean
```

A terminal window titled 'pi@raspberrypi: ~' with standard window controls. The terminal shows the command 'make clean' being executed. The output is 'rm helloworld', followed by a new prompt 'pi@raspberrypi ~ \$' with a green cursor.

```
pi@raspberrypi ~ $ make clean
rm helloworld
pi@raspberrypi ~ $
```

실습 3-1: make

14

- Makefile 수정

```
helloworld : helloworld.c
    gcc -Wall -W -o helloworld helloworld.c

clean :
    rm helloworld

hi :
    echo hello~

ping :
    echo pong

list :
    ls -al *.c

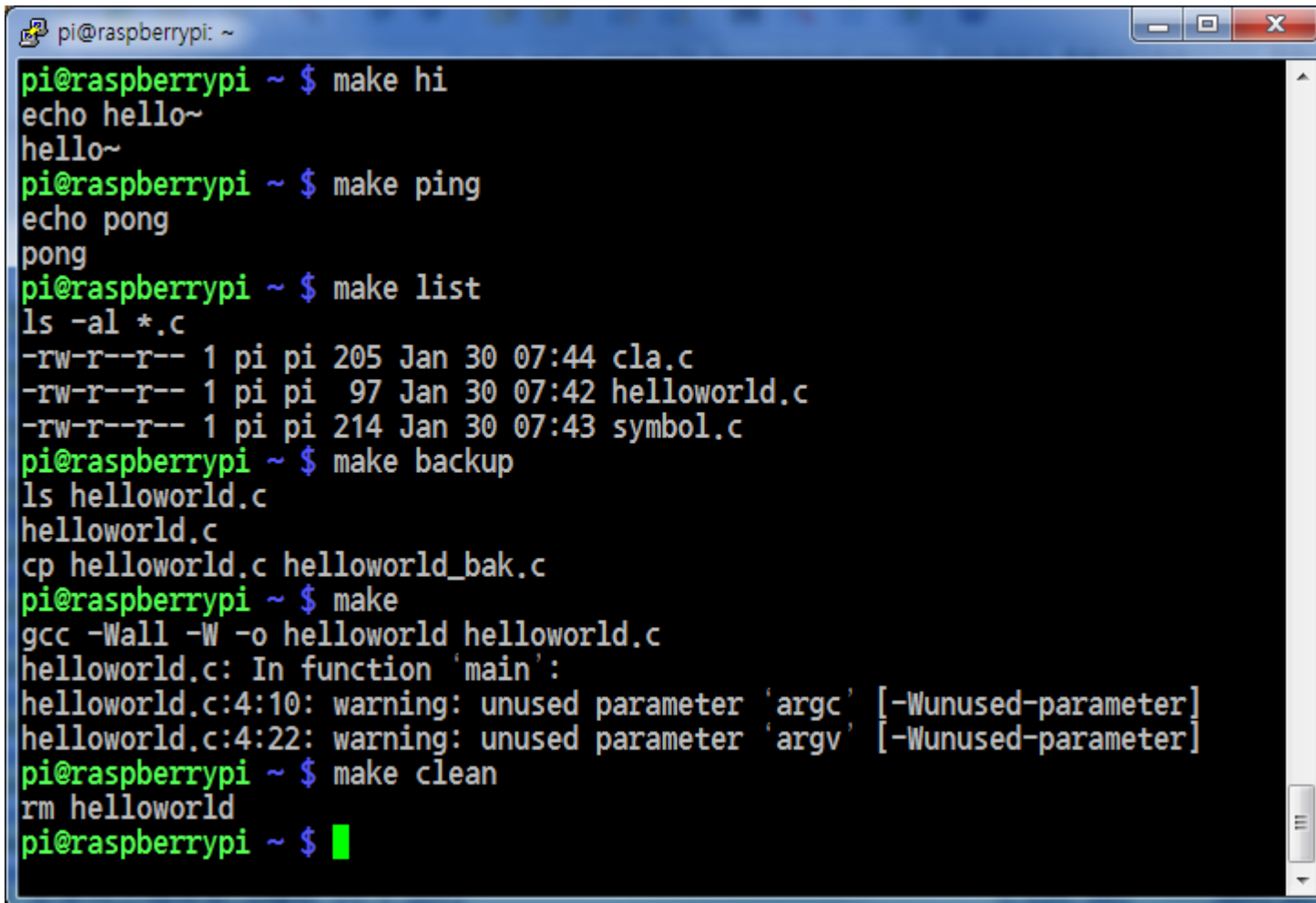
backup :
    ls helloworld.c
    cp helloworld.c helloworld_bak.c

everything : hi ping list backup
```

실습 3-2: make

15

- 타겟 실행

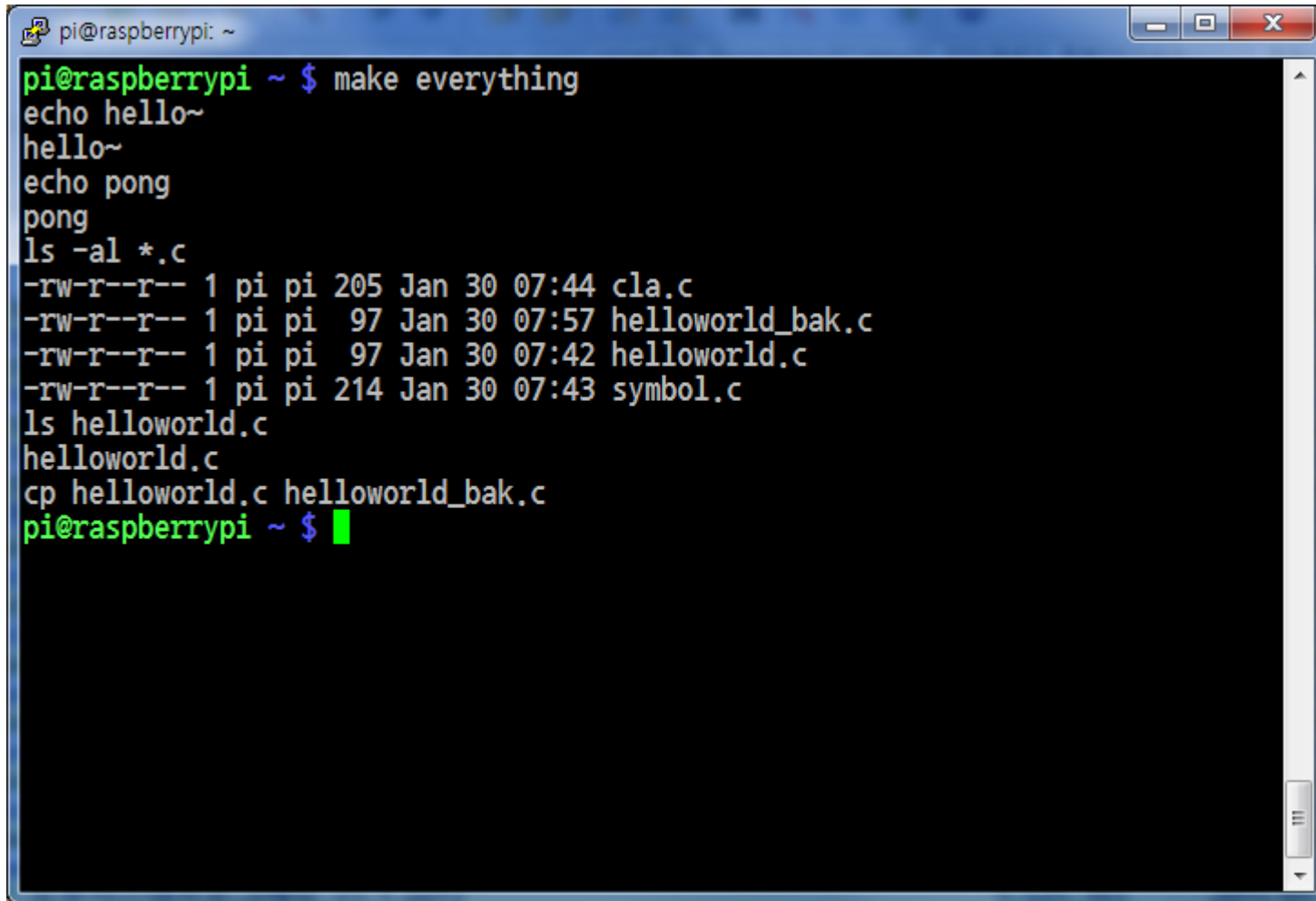


```
pi@raspberrypi: ~  
pi@raspberrypi ~ $ make hi  
echo hello~  
hello~  
pi@raspberrypi ~ $ make ping  
echo pong  
pong  
pi@raspberrypi ~ $ make list  
ls -al *.c  
-rw-r--r-- 1 pi pi 205 Jan 30 07:44 cla.c  
-rw-r--r-- 1 pi pi 97 Jan 30 07:42 helloworld.c  
-rw-r--r-- 1 pi pi 214 Jan 30 07:43 symbol.c  
pi@raspberrypi ~ $ make backup  
ls helloworld.c  
helloworld.c  
cp helloworld.c helloworld_bak.c  
pi@raspberrypi ~ $ make  
gcc -Wall -W -o helloworld helloworld.c  
helloworld.c: In function 'main':  
helloworld.c:4:10: warning: unused parameter 'argc' [-Wunused-parameter]  
helloworld.c:4:22: warning: unused parameter 'argv' [-Wunused-parameter]  
pi@raspberrypi ~ $ make clean  
rm helloworld  
pi@raspberrypi ~ $
```

실습 3-3: make

16

- 타겟 실행



```
pi@raspberrypi: ~  
pi@raspberrypi ~ $ make everything  
echo hello~  
hello~  
echo pong  
pong  
ls -al *.c  
-rw-r--r-- 1 pi pi 205 Jan 30 07:44 cla.c  
-rw-r--r-- 1 pi pi 97 Jan 30 07:57 helloworld_bak.c  
-rw-r--r-- 1 pi pi 97 Jan 30 07:42 helloworld.c  
-rw-r--r-- 1 pi pi 214 Jan 30 07:43 symbol.c  
ls helloworld.c  
helloworld.c  
cp helloworld.c helloworld_bak.c  
pi@raspberrypi ~ $
```


실습 4-1: make

17

```
#include<stdio.h>
```

파일명 : main.c

```
void choi_f1();
```

```
void choi_f2();
```

```
void kim_f1();
```

```
void kim_f2();
```

```
void lee_f1();
```

```
extern int i;
```

```
int main(void)
```

```
{
```

```
    choi_f1();
```

```
    choi_f2();
```

```
    kim_f1();
```

```
    kim_f2();
```

```
    lee_f1();
```

```
    printf("[%s] in %s:%d\n", __FILE__, __func__, __LINE__);
```

```
    printf("i = %d\n", i);
```

```
    return 0;
```

```
}
```

실습 4-2: make

18

```
#include<stdio.h>
```

파일명 : choi.c

```
int i = 10;
```

```
void choi_f1()
```

```
{
```

```
    printf("[%s] in %s:%d\n", __FILE__, __func__, __LINE__);
```

```
}
```

```
void choi_f2()
```

```
{
```

```
    printf("[%s] in %s:%d\n", __FILE__, __func__, __LINE__);
```

```
}
```

실습 4-3: make

19

```
#include<stdio.h>
```

파일명 : lee.c

```
static void lee_f2();
```

```
void lee_f1()
```

```
{
```

```
    printf("[%s] in %s:%d\n", __FILE__, __func__, __LINE__);  
    lee_f2();
```

```
}
```

```
static void lee_f2()
```

```
{
```

```
    printf("[%s] in %s:%d\n", __FILE__, __func__, __LINE__);
```

```
}
```

실습 4-4: make

20

```
#include<stdio.h>
```

파일명 : kim.c

```
void kim_f1()  
{  
    printf("[%s] in %s:%d\n",__FILE__,__func__,__LINE__);  
}
```

```
void kim_f2()  
{  
    printf("[%s] in %s:%d\n",__FILE__,__func__,__LINE__);  
}
```

실습 4-5: make

21

```
~$ gcc -Wall -W -c lee.c
~$ gcc -Wall -W -c kim.c
~$ gcc -Wall -W -c choi.c
~$ gcc -Wall -W -c main.c
~$ gcc -Wall -W lee.o kim.o choi.o main.o -o prog
~$ ./prog
[choi.c] in choi_f1:7
[kim.c] in kim_f1:5
[lee.c] in lee_f1:7
[lee.c] in lee_f2:13
[choi.c] in choi_f2:12
[kim.c] in kim_f2:10
[main.c] in main:19
i = 10
~$
```

실습 5: make

22

- Makefile

```
prog : main.o choi.o kim.o lee.o
    gcc -Wall -W main.o choi.o kim.o lee.o -o prog

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

choi.o : choi.c
    gcc -Wall -W -c choi.c

kim.o : kim.c
    gcc -Wall -W -c kim.c

lee.o : lee.c
    gcc -Wall -W -c lee.c
```

매크로 정의

23

- **매크로 정의**
 - NAME = string
- **매크로 참조**
 - \$을 시작으로 괄호 또는 중괄호
ex) \${NAME} or \$(NAME)
- **정의 되지 않은 매크로를 참조할 때는 null 문자열로 치환됨**
- **중복된 정의는 마지막에 정의된 값을 사용**
- **매크로 정의 시 이전에 정의된 매크로를 참조를 통해 정의 가능**
 - NAME2 = my \$(NAME)

매크로 정의 시 주의 사항

24

- 문자열에 따옴표를 넣으면 따옴표 또한 문자열의 일부로 인식됨
- 매크로의 이름에 ':', '=', '#' 을 사용 할 수 없음
- Tab으로 시작하면 안됨
- 매크로는 반드시 사용될 위치보다 먼저 정의 되어야 함

실습 6 : make

25

```
OBJECTS = main.o choi.o kim.o lee.o
TARGET = prog
CC = gcc
CFLAGS = -Wall -W

$(TARGET) : $(OBJECTS)
    $(CC) $(CFLAGS) -o $(TARGET) $(OBJECTS)

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

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

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

lee.o : lee.c
    $(CC) $(CFLAGS) -c lee.c
```

미리 정의된 매크로

26

- 미리 정의된 매크로
- Makefile 실습

```
all:
    @echo SHELL = $(SHELL)
    @echo CC = $(CC)
    @echo LD = $(LD)
    @echo AS = $(AS)
    @echo AR = $(AR)
    @echo PWD = $(PWD)
    @echo HOME = $(HOME)
    @echo CXX = $(CXX)
```

자동 매크로

27

• 자동 매크로 리스트

매크로	값
\$?	타겟보다 최근에 변경된 종속 항목 리스트 (확장자 규칙에서 사용 불가)
\$^	타겟의 종속 항목 리스트 (확장자 규칙에서 사용 불가)
\$@	타겟의 이름
\${@F}	타겟의 파일 부분
\${@D}	타겟의 경로 부분
\$<	타겟보다 최근에 변경된 종속 항목 리스트 (확장자 규칙에서만 사용 가능)
\$*	타겟보다 최근에 변경된 종속 항목 리스트(확장자 제외) (확장자 규칙에서만 사용 가능)

실습 7: make

28

```
OBJECTS = main.o choi.o kim.o lee.o
TARGET = prog
CC = gcc
CFLAGS = -Wall -W

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

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

choi.o : choi.c
    $(CC) $(CFLAGS) -c $^

kim.o : kim.c
    $(CC) $(CFLAGS) -c $^

lee.o : lee.c
    $(CC) $(CFLAGS) -c $^

clean :
    rm $(TARGET)
    rm $(OBJECTS)
```

실습 8: make

29

```
OBJECTS = main.o choi.o kim.o lee.o
TARGET = prog
CC = gcc
CFLAGS = -Wall -W

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

clean :
    rm $(TARGET)
    rm $(OBJECTS)
```

실습 9: 명령(Command)

30

```
myfile :  
        cd ~  
        ls -al *
```

```
myfile :  
        cd ~ ; ls -al *
```

```
myfile :  
        cd /abcd ; ls -al *
```

```
myfile :  
        cd /abcd && ls -al *
```

실습 10: 명령(Command)

31

```
view :  
    ls file.c  
    cat file.c  
    echo complete!
```

```
view :  
    -ls file.c  
    -cat file.c  
    @echo complete!
```

실습 11: 특수 타겟

32

- .SILENT:
- .IGNORE:

```
everything : hi ping list

hi:
    echo hi

ping:
    echo pong

list:
    ls -al

test:
    ls test.txt
    cp test.txt test_bak.txt
```

. IGNORE:

. SILENT:

참고 자료 1 (파일의 시간정보)

33

- **마지막 접근 시간(ls -lu)**
 - stat구조체의 st_atime
 - read 연산
- **마지막 수정 시간(ls -l)**
 - stat구조체의 st_mtime
 - write 연산
- **i-node 상태의 마지막 수정 시간(ls -lc)**
 - stat 구조체의 st_ctime
 - chmod, chown, link 등등

```
struct stat {
    mode_t st_mode;
    ino_t st_ino;
    dev_t st_dev;
    dev_t st_rdev;
    nlink_t st_nlink;
    uid_t st_uid;
    gid_t st_gid;
    off_t st_size;
    time_t st_atime;
    time_t st_mtime;
    time_t st_ctime;
    long st_blk_size;
    long st_blocks;
};
```

참고 자료 1 (utime)

34

```
#include <sys/types.h>
#include <utime.h>

int utime(const char *pathname, const struct utimbuf *times );
```

- **기능: 파일의 시간 설정**
- **리턴 값: 성공하면 0, 실패하면 -1**
- **utimbuf 구조체**

```
struct utimbuf {
    time_t  actime;          /* 마지막 접근 시간 */
    time_t  modtime;         /* 마지막 수정 시간 */
}
```

- 각 필드는 1970. 1.1. 00:00 부터 현재까지의 시간을 초로 환산한 값
- times가 NULL 이면, 현재시간으로 설정됨

참고 자료 1 (utime 예제)

35

```
#include<stdio.h>
#include<fcntl.h>
#include<utime.h>
#include<sys/stat.h>
#include<unistd.h>

int main(int argc, char *argv[])
{
    int i, fd;
    struct stat statbuf;
    struct utimbuf timebuf;

    for (i = 1; i < argc; i++) {
        if (stat(argv[i], &statbuf) < 0) { /* fetch current times */
            printf("%s: stat error", argv[i]);
            continue;
        }

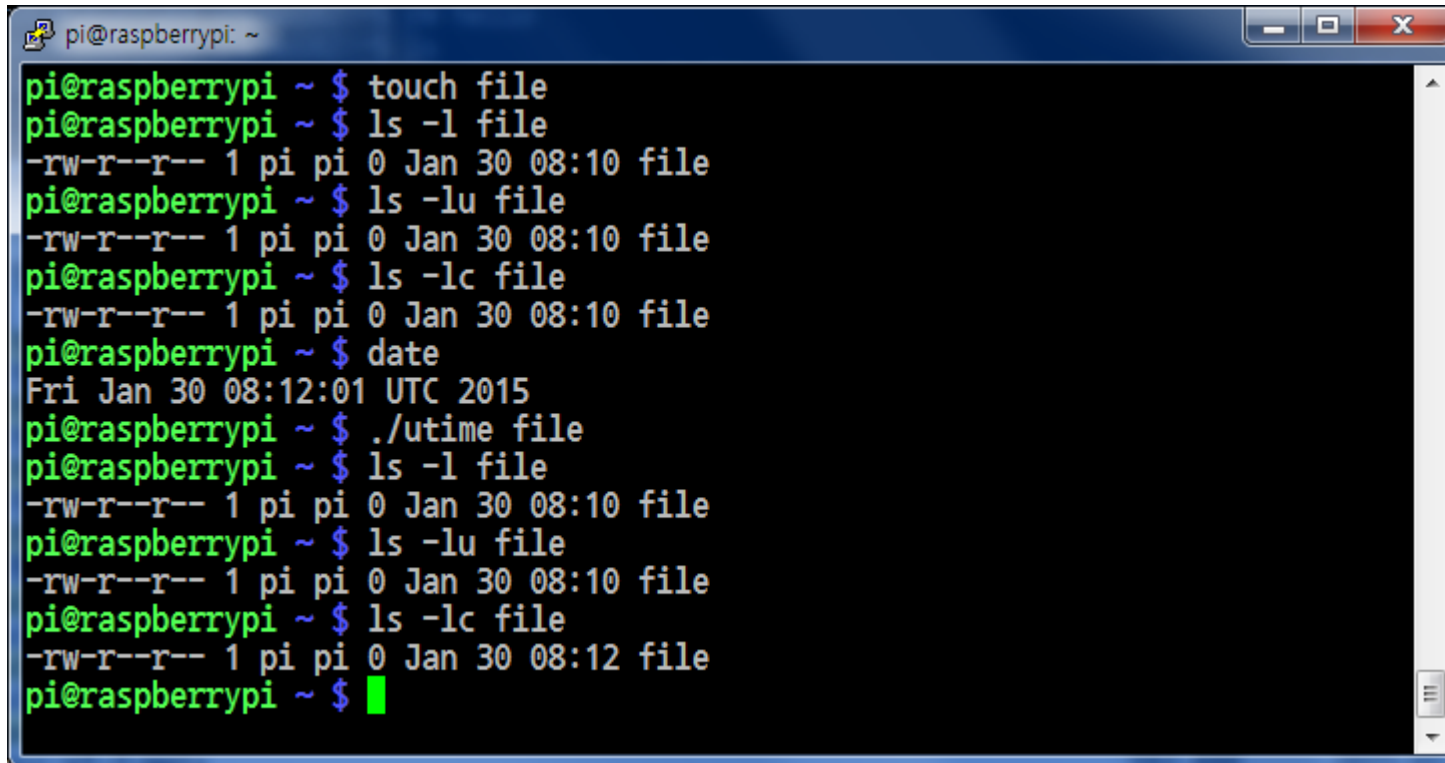
        if ((fd = open(argv[i], O_RDWR | O_TRUNC)) < 0) { /* truncate
*/
            printf("%s: open error", argv[i]);
            continue;
        }
        close(fd);

        timebuf.actime = statbuf.st_atime;
        timebuf.modtime = statbuf.st_mtime;
    }
}
```

참고 자료 1 (utime 예제)

36

```
if (utime(argv[i], &timebuf) < 0) {      /* reset times */
    printf("%s: utime error", argv[i]);
    continue;
}
return 0;
}
```



```
pi@raspberrypi: ~
pi@raspberrypi ~ $ touch file
pi@raspberrypi ~ $ ls -l file
-rw-r--r-- 1 pi pi 0 Jan 30 08:10 file
pi@raspberrypi ~ $ ls -lu file
-rw-r--r-- 1 pi pi 0 Jan 30 08:10 file
pi@raspberrypi ~ $ ls -lc file
-rw-r--r-- 1 pi pi 0 Jan 30 08:10 file
pi@raspberrypi ~ $ date
Fri Jan 30 08:12:01 UTC 2015
pi@raspberrypi ~ $ ./utime file
pi@raspberrypi ~ $ ls -l file
-rw-r--r-- 1 pi pi 0 Jan 30 08:10 file
pi@raspberrypi ~ $ ls -lu file
-rw-r--r-- 1 pi pi 0 Jan 30 08:10 file
pi@raspberrypi ~ $ ls -lc file
-rw-r--r-- 1 pi pi 0 Jan 30 08:12 file
pi@raspberrypi ~ $
```

참고 자료 2 (main함수의 반환)

37

- main 함수
 - 반환 값 확인
 - \$ echo \$?
 - 반환 값 의미
 - 성공 시 0
 - 실패 시 0이 아닌 값
 - &&
 - \$./hello1 && ./hello2
 - \$./hello2 && ./hello1
 - 실습

#include<stdio.h>

hello1.c

```
int main(int argc, char *argv[])
{
    printf("hello world 1\n");
    return 0;
}
```

#include<stdio.h>

hello2.c

```
int main(int argc, char *argv[])
{
    printf("hello world 2\n");
    return 1;
}
```

참고 자료 3 - 확장자 규칙

38

- 컴파일러 -c 옵션으로 컴파일 하면 그에 대응하는 *.o 파일이 만들어짐
 ※ C언어 소스파일은 *.c 확장자로 되어 있어야 함(필수)
- 확장자
 - C : .c
 - JAVA : .java
 - C++ : .cc
 - 포트란 : .f
- 확장자가 갖는 규칙에 기초해 명령을 알아서 해석, 자동화
- 내부적으로 정의되어 있는 확장자 규칙
 \$ make -p

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

미션

39

- GPIO 12 LED**연결**
- **타겟**
 - init : GPIO 12 을 OUTPUT **모드로 변경하고 0으로 초기화**
 - on : LED ON
 - off : LED OFF
 - toggle : **1초 단위로** led on / off **10회**
- Ex)
 - \$ make init
 - \$ make on
 - \$ make off
 - \$ make toggle