Part 01. 리눅스 개발 환경

Chapter 05. make 및 Makefile

진행 순서

Chapter 05_01 make 개요

make는 소프트웨어 개발을 위해 리눅스 운영 체제에서 주로 사용되는 프로그램 빌드 도구이다.

여러 파일들끼리의 의존성과 각 파일에 필요한 명령을 정의함으로써 프로그램을 컴파일할 수 있으며 최종 프로그램을 만들 수 있는 과정을 서술할 수 있는 표준적인 문법을 가지고 있다.

위의 구조로 기술된 파일(주로 Makefile)을 make가 해석하여 프로그램 빌드를 수행하게 된다.

make 설치

```
[root@localhost ~]# yum install make
```

Chapter 05_02 Makefile

make 프로그램이 작동되면서 기본적으로 사용되는 파일의 이름이 Makefile

Makefile 속에 관리하고자 하는 소스 파일과 헤더 파일, 컴파일 옵션, 링크 옵션 등 필요한 내용들을 서술한다.
다음 make를 실행하면서 Makefile을 읽어 들여 내용을 수행한다.

make 프로그램 실행

```
[root@localhost ~]# make
```

Makefile이 아닌 다른 파일 명 지정

```
[root@localhost ~]# make  –f  [파일명]
```

Chapter 05_02 Makefile

단순한 **Makefile**은 다음 형태의 규칙들로 구성되어 있다.

```
target ... : prerequisites ...
        recipe
        ...
        ...
```

target(대상)은 일반적으로 프로그램에 의해 생성되는 파일의 이름으로
실행 파일 또는 **object** 파일입니다.

prerequisites(전제 조건)은 **target**을 작성하기 위한 입력으로 사용되는 파일입니다.

recipe(레시피)는 수행하는 동작을 나타냅니다.

참고: 모든 **recipe** 줄의 시작 부분에 탭 문자를 넣어야 합니다.

Chapter 05_02 Makefile

```
edit : main.o kbd.o command.o display.o \
       insert.o search.o files.o utils.o
                cc -o edit main.o kbd.o command.o display.o \
                   insert.o search.o files.o utils.o

main.o : main.c defs.h
                cc -c main.c
kbd.o : kbd.c defs.h command.h
                cc -c kbd.c
command.o : command.c defs.h command.h
                cc -c command.c
display.o : display.c defs.h buffer.h
                cc -c display.c
insert.o : insert.c defs.h buffer.h
                cc -c insert.c
search.o : search.c defs.h buffer.h
                cc -c search.c
files.o : files.c defs.h buffer.h command.h
                cc -c files.c
utils.o : utils.c defs.h
                cc -c utils.c
clean :
                rm edit main.o kbd.o command.o display.o \
                   insert.o search.o files.o utils.o
```

간단한 Makefile 예제
(https://www.gnu.org/software/make/manual/make.html)

edit 이라는 실행 파일이 8개의 object 파일에 의존하고,
8개의 소스 파일과 3개의 헤더 파일에 의존하는 방식

clean 대상은 파일이 아니라 행동의 이름
전제 조건 없이 레시피 실행
일반적으로 컴파일 된 결과를 정리할 때 사용하는 clean

Chapter 05_02 Makefile

```
CFLAGS = -g –O2 -I../include
LDLIBS = -lpthread

TARGET = test_program
OBJS = main.o config.o debug.o log.o

all : $(OBJS)
        cc –o $(TARGET) $(OBJS) $(LDLIBS)

clean :
        rm –f $(OBJS)
        rm –f $(TARGET)
```

Makefile 실사용예

CFLAGS, LDFLAGS, TARGET, OBJS 등과 같은 매크로 지정

test_program : main.o config.o debug.o log.o
구문을 all : $(OBJS) 처럼 간단하게 표현 가능
중복된 구문이 많을 경우 편리하게 사용

make
- 타겟이 없을 경우 암묵적으로 all 실행

make all
- 명시적으로 all 타겟 지정, 다른 타겟이 존재할 경우 make [타겟] 과 같이 지정 가능

make clean
- clean 레이블 실행, 바이너리 및 오브젝트 파일 정리

Chapter 05_03 매크로

```
OBJS = main.o  config.o  debug.o  log.o
...
all : $(OBJS)
        cc  –o  $(TARGET)  $(OBJS)  $(LDLIBS)
```

매크로는 변수와 같이 이름 선언하고 내용 서술
ex) 매크로=...
$(매크로명)를 이용하여 작성한 매크로 사용

Makefile 내에 매크로를 활용하면 중복적으로 표현되는 구문을 방지하고
특정 파일 이름 수정 시 모든 내용을 찾아서 수정할 필요가 없다.

Chapter 05_04 주요 사전 정의 변수

CC : C 컴파일러, 기본값 cc
CFLAGS : 컴파일 옵션
ex) CFLAGS = -g –O2 –I../includes
LDFLAGS : 컴파일러가 -L과 같은 링커 'ld'를 호출 할 때 제공 할 추가 플래그입니다.
라이브러리 (ex. -lfoo)를 LDLIBS 변수에 추가해야 합니다.
ex) LDFLAGS = -L. –L$(LIB_DIR)
LDLIBS : 링커 'ld'를 호출 할 때 컴파일러에 제공되는 라이브러리 플래그 또는 이름입니다.
-L과 같은 비 라이브러리 링커 플래그는 LDFLAGS 변수에 있어야 합니다.
ex) LDLIBS = -lpthread

Chapter 05_05 주요 자동 변수

$@
- 규칙 대상의 파일 이름
- 대상이 여러 개인 패턴 규칙에서 규칙의 레시피가 실행된 대상의 이름

$<
- 첫 번째 전제 조건의 이름

$?
- 타겟보다 새로운(변경된 파일) 모든 전제 조건의 이름

$^
- 모든 전제 조건의 이름

$*
- 타겟 이름이 확장자로 끝나는 경우, 확장자를 뺀 이름
- ex) 타겟 이름이 'foo.c'인 경우 '$*'는 'foo'로 설정

```
CFLAGS = -Wall
TARGET = test_program
OBJS = main.o sub.o

all : $(TARGET)

$(TARGET) : $(OBJS)
		$(CC) $(CFLAGS) –o $@ $^

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

sub.o : sub.c
		$(CC) $(CFLAGS) –c –o $@ $^

clean :
		rm –rf *.o $(TARGET)
```

Chapter 05_06 패턴 규칙

접미사 규칙(확장자 규칙)은 구식 방법
패턴 규칙이 더 일반적이고 명확하기 때문에 접미사 규칙은 더 이상 사용되지 않습니다.
이전 makefile과의 호환성을 위하여 GNU make에서 지원
'%.c' 패턴은 '.c'로 끝나는 모든 파일 이름과 일치
's.%.c' 패턴은 's.'으로 시작하고 '.c'로 끝나는 파일 이름과 일치('%'와 일치하는 문자 하나 이상 존재)

이전 접미사 규칙 방식

```
.c.o:
        $(CC) $(CFLAGS) –c $< –o $@
```

패턴 규칙 방식

```
%.o : %.c
        $(CC) $(CFLAGS) –c $< –o $@
```

'.c' 파일을 '.o' 파일로 컴파일하는 규칙

Chapter 05_07 치환 매크로

치환 매크로를 사용하면 반복적인 파일의 나열을 편리하게 특정 파일로 치환하여 사용할 수 있다.
$(매크로:변경전=변경후)
변경전이 변경후로 바뀌어 사용된다.

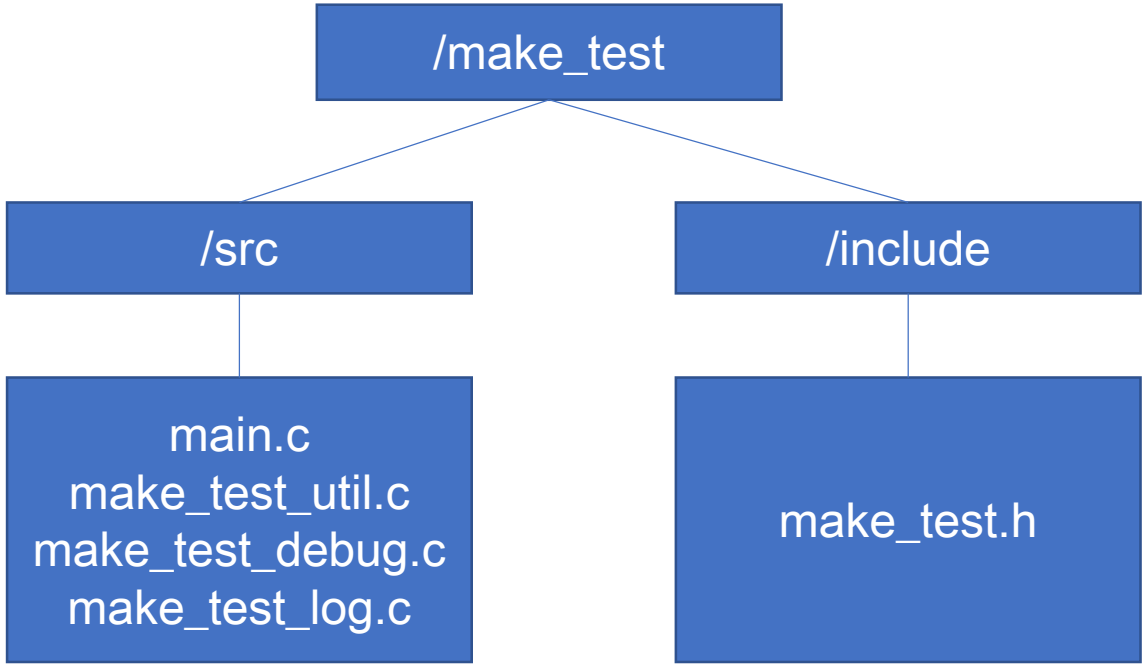
```
SRCS = main.c  command.c  display.c  utils.c  files.c
OBJS = main.o  command.o  display.o  utils.o  files.o
```

상기와 같이 반복적으로 작성되는 파일 이름을 아래와 같이 치환 매크로를 이용할 수 있다.

```
SRCS = main.c  command.c  display.c  utils.c  files.c
OBJS = $(SRCS:.c=.o)
```

Chapter 05_08 make 실습

소스코드 디렉토리 구조




```
[root@localhost ]# ls
include  src
[root@localhost ]# ls include/
make_test.h
[root@localhost ]# ls src/
Makefile  main.c  make_test_debug.c  make_test_log.c  make_test_util.c
```

Chapter 05_08 make 실습

make_test.h

```
#ifndef __MAKE_TEST_H__
#define __MAKE_TEST_H__

#define MAX_NUM 5

#endif
```

main.c

```
#include <stdio.h>
#include <pthread.h>
#include "make_test.h"

void *make_test_util(void *arg);
void *make_test_debug(void *arg);
void *make_test_log(void *arg);

int main()
{
        pthread_t thread_util;
        pthread_t thread_debug;
        pthread_t thread_log;

        int *thread_util_ret;
        int *thread_debug_ret;
        int *thread_log_ret;

        printf("main start\n");
        printf("MAX NUM is %d\n", MAX_NUM);

        pthread_create(&thread_util, NULL, make_test_util, NULL);
        pthread_create(&thread_debug, NULL, make_test_debug, NULL);
        pthread_create(&thread_log, NULL, make_test_log, NULL);

        pthread_join(thread_util, (void **)&thread_util_ret);
        pthread_join(thread_debug, (void **)&thread_debug_ret);
        pthread_join(thread_log, (void **)&thread_log_ret);

        return 0;
}
```

Chapter 05_08 make 실습

make_test_util.c

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include "make_test.h"

void *make_test_util(void *arg)
{
	int i;
	for (i = 0; i < MAX_NUM; i++) {
		printf("(%d) make_test_util\n", i+1);
		sleep(1);
	}
	pthread_exit((void *)1);
}
```

make_test_log.c

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include "make_test.h"

void *make_test_log(void *arg)
{
	int i;
	for (i = 0; i < MAX_NUM; i++) {
		printf("(%d) make_test_log\n", i+1);
		sleep(1);
	}
	pthread_exit((void *)1);
}
```

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include "make_test.h"

void *make_test_debug(void *arg)
{
	int i;
	for (i = 0; i < MAX_NUM; i++) {
		printf("(%d) make_test_debug\n", i+1);
		sleep(1);
	}
	pthread_exit((void *)1);
}
```

make_test_debug.c

Chapter 05_08
make 실습

Makefile

```
CC = gcc
CFLAGS = -g -Wall -I../include
LDLIBS = -lpthread
TARGET = make_test
SRCS = main.c make_test_util.c make_test_debug.c make_test_log.c
OBJS = $(SRCS:.c=.o)

all: $(TARGET)

$(TARGET): $(OBJS)
	$(CC) $(CFLAGS) -o $@ $^ $(LDLIBS)

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

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

```
[root@localhost src]# ls
Makefile  main.c  make_test_debug.c  make_test_log.c  make_test_util.c
[root@localhost src]# make
gcc -g -Wall -I../include -c main.c -o main.o
gcc -g -Wall -I../include -c make_test_util.c -o make_test_util.o
gcc -g -Wall -I../include -c make_test_debug.c -o make_test_debug.o
gcc -g -Wall -I../include -c make_test_log.c -o make_test_log.o
gcc -g -Wall -I../include -o make_test main.o make_test_util.o make_test_debug.o make_test_log.o -lpthread
[root@localhost src]# ls
Makefile  main.o    make_test_debug.c  make_test_log.c  make_test_util.c
main.c    make_test  make_test_debug.o  make_test_log.o  make_test_util.o
```