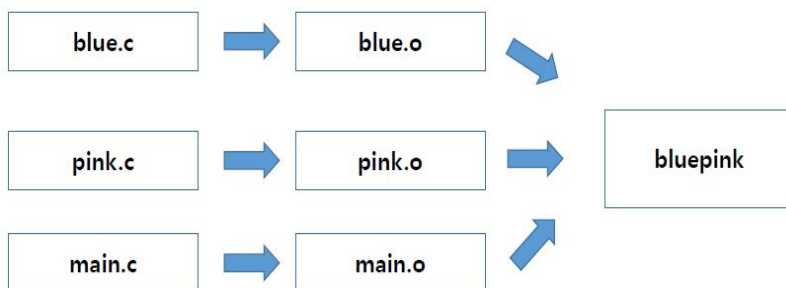


# gcc Build Process

## makefile을 왜 사용 할까?

- gcc 빌드의 경우 object 파일을 생성하고 만들어진 object 파일+ Library를 모아 하나로 합침  
-> 하지만 매번 C파일을 갱신할때 기존의 모든 object 파일 삭제 및 다시 생성

```
gcc c ./ blue.c
gcc c ./ pink c
gcc c ./main.c
gcc ./main.o blue.o pink.o o ./ bluepink
```



makefile을 사용하면 변경된 부분만 한번에 갱신이 가능하다!

## makefile 설치 및 실행하기

- 설치

---

sudo apt install make -y    makefile 생성시 필요 명령어

---

sudo apt install g++ -y    c++파일 컴파일러 이지만 makefile 실행시 필요한듯

- 파일 생성

---

vi makefile	makefile의 스펠링을 변경되면 안됨! 단(대소문자 구별은 안함) 실제 "Makefile"로도 많이 생성
-------------	---

---

make	makefile 실행
------	-------------

## makefile 작성하기

---

rm -r ./*.o	현재 dirct에 있는 전체 object파일 삭제(띄어쓰기 주의할것)
-------------	--

---

rm ./[object 파일명]	object 파일명
-------------------	------------

## cmake

cmake 란 대부분의 플랫폼에서 빌드를 쉽게 할 수있도록 도와주는 프로그램  
ex) vs코드에서 짠 코드를  
리눅스에서도 돌려보고 싶다? -> 원래는 프로젝트 생성 Lib 추가 v파일추가 등등.  
cmake 파일을 이용하면 바로 가능!

- 설치 명령어:  
sudo apt install g++ cmake -y
- 1. 실행할 C 파일 생성
- 2. vi CMakeLists.txt 을 실행하여 CMakeLists.txt 생성(이름 철자 주의!)
- 3. CMakeLists.txt 안에 예: ADD\_EXECUTABLE([실행프로그램이름] [실행코드1] [실행코드2] ...) 작성  
ex) ADD\_EXECUTABLE(gogo test.c main.c)
- 4. cmake . 입력
- 5. makefile이 자동으로 생성된것을 확인
- 6. make 입력 후 ./[실행프로그램이름] 입력
- 7. 정상 출력 확인

make 스크립트

왜 사용할까

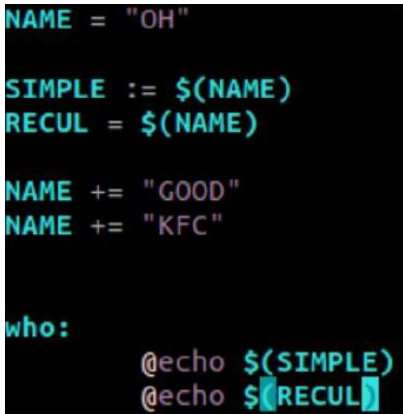
기본 구조

- 1. vi makefile실행
- 2. makefile 입력 ex)

```
1 HI:
2     echo "HI";
3 HO:
4     echo "hay";
```

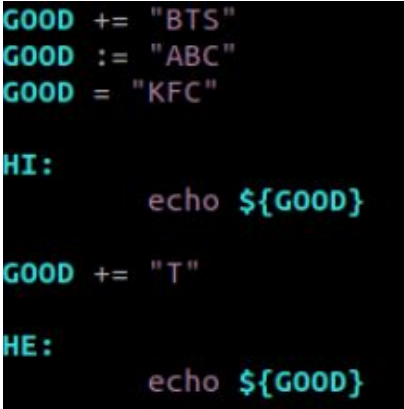
- HI , HO: target 역할
- echo 화면 출력 shell 명령어  
(makefile의 printf같은 느낌 ?)
- 3. 실행 방법 두가지
  - make (첫번째 target을 수행)
  - make HO (지정된 target수행)

명령어(코드)	의미	예시
#	주석처리	#주석입니다.
@echo "문자열";	echo출력 안됨, 문자열만 출력	@echo "hello";
의존성타겟: 타겟 타겟	HA를 실행하면 HI HO의 내용 출력(HA는 의존성 타겟)	HA: HI HO

명령어(코드)	의미	예시
매크로이름 = "문자열"	매크로 이름과 내용 설정	MSG = "test"
@echo \$(MSG);	MSG매크로 출력 (띄어쓰기 주의!!)	@echo \$(MSG);
매크로 += "문자열"	기존 매크로 뒤에 내용이 붙음(자동 띄어쓰기가 됨)	MSG += "yellow"
매크로 := \$(기존 매크로)	현재까지 저장된 기존 매크로가 출력됨	 <pre> NAME = "OH" SIMPLE := \$(NAME) RECU = \$(NAME) NAME += "GOOD" NAME += "KFC"  who:     @echo \$(SIMPLE)     @echo \$\$(RECU) </pre> <p>SIMPLE 출력: OH , RECU 출력: OH GOOD KFC</p>
\$@	현재 자신의 target이름 출력( 오른쪽 출력 결과 ok)	ok: @echo "\$@"

단! 매크로의 경우 맨위에 써주는 것이 가독성이 좋다.

- 이유:



```

GOOD += "BTS"
GOOD := "ABC"
GOOD = "KFC"

HI:
    echo ${GOOD}

GOOD += "T"

HE:
    echo ${GOOD}

```

출력: HI의 경우 BTS T , HE의 경우도 BTS T

이유: "=" 이후로 ":=" 이없다면 아래 있는 매크로까지 전부 계산후 타겟안으로 들어간다.  
따라서 맨위에 전부쓰는것과 동일하다고 생각하자.

자동화 스크립트

```
A :
    mkdir hi
    cd hi
    ls

B :
    cd ..
    rmdir hi
    ls
```

dd 명령어를 이용한 dummy data 만들기

- dd 명령어 역할
  1. 파일 복사
  2. 파일 읽기
  3. 하드디스크 통채로 복사
  4. (중요!) 성능테스트 사용시

**입력 파일: if(inputfile), 아웃풋 파일: of(outputfile)**  
**bs(blocksize), count** 몇번의 입출력을 수행할지 의미. 즉 파일의 크기는 **bs+count**  
**예시) bs =1000M count = 1000** 이라하면 총1GB가 생성됨,

dd if=/dev/zero of=test.bin bs=1024 count=1	파일 쓰기
dd if=test.bin of=/dev/zero bs=1024	파일읽기(count가 읽기경우 파일 전체를 읽는다.)

-> if of가 서로 바뀐것을 확인 할 수 있다.



- 하지만 서버의 경우 하드디스크 보다는 ram을 더 이용을 많이 한다.  
따라서 ram의 성능 측정을 위해서는

```
cd /dev/shm
```

을 입력한후 위의 코드를 입력하면 된다.

shell 명령어

cat	파일 내용을 출력한다 뒤에 "> txt파일" 을 붙이면 내용을 txt파일로 저장한다.	\$cat /proc/cpuinfo \$cat /proc/cpuinfo>bts.txt
grep	파일을 찾는 명령어	

find [ 경로 ] name "파일명"	<div>- 파일을 찾는 명령어 (. / 는 폴더고 /는 디렉토리)</div> <div>- **안의 이름이 포함되어있는 모든 파일 디렉토리 탐색</div> <div>( *가 이름 앞이면 확장자 *가 뒤면 파일명 앞뒤 있으면 둘다)</div> <div>- 파일혹은 디렉토리만 검색 (뒤에 -type f or d)를 붙임</div>	<div>\$find ./dir -name "kfc.txt"</div> <div>\$find -name <b>"*kfc*"</b></div> <div>\$find -name "kfc.txt" -type f</div>
history	모든 명령어 입력했던것들을 보여줌	
du -sh ./디렉토리	디렉토리 용량 확인 뒤에 안붙이면 현재 디렉토리 용량	
\$ls -alh ./a.log	파일용량 확인은 이걸로인듯?	
file [이름]	이름적은 것이 무슨type인지 확인	
grep	<div><div>• 문자열 검색</div><div>• 내용에 문자열을 검색하고 싶을때 사용한다.<div>• 정규표현식 사용 가능 (정규표현식은 차후 학습 예정)</div></div><div>• meminfo 파일 내 "kB" 문자열 검색<div>• <b>\$grep kB ./meminfo</b></div><div>• kB 문자열이 포함된 한 줄 라인이 출력된다.</div></div><div>• 현재 디렉토리에 있는 모든 파일을 대상으로, "ache" 가 들어있는 라인 모두 출력<div>• <b>\$grep ache ./ * -r</b></div><div>• -r 옵션 : 내부 디렉토리에 있는 파일들도 모두 찾는다.</div></div><div>• 다른 명령어 출력결과에 문자열 검색하기 (bar를 파이프라고 부름)<div>• <b>\$ls -al   grep .v</b></div><div>• ls -al 출력 결과를 오른쪽으로 넘겨서 처리하는 원리 예시. \$cat abc.txt   grep bbq</div></div></div> <div></div>	

tree (설치해야함)	자신과 하위디렉토리들을 손쉽게 볼수있음
파일 압축 및 풀기	

## · 아래의 명령어를 입력한 후, 압축 파일의 용량을 확인하자

- 1. `$zip all1.zip a.log`
  - -> all1.zip 용량 확인
- 2. all1.zip 삭제 후, `$zip all2.zip ./*`
  - -> all2.zip 용량 확인 후, GUI Shell 로 all2.zip 열어서 내용 확인하기
- 2. all2.zip 삭제 후, `$zip -r all3.zip ./*`
  - -> all3.zip 용량 확인 후, GUI Shell 로 all3.zip 열어서 내용 확인하기

## unzip 명령어 사용

- **unzip [압축 파일명]**
  - 현재 디렉토리로 압축을 푼다.
  - `$unzip all3.zip -> a.log / b.log` 가 나온다. `unzip_a.log`, `unzip_b.log` 로 각각 rename 하자
- **unzip [압축 파일명] -d [디렉토리]**
  - 특정 디렉토리로 압축을 푼다.
  - `$unzip all3.zip -d ./tt ->` 현재 디렉토리에 ./tt 디렉토리 생성 후 그 안에 압축해제

### 다른 방법

<code>gzip [파일명]</code>	파일 압축
<code>gunzip [파일명]</code>	압축 풀기
<code>xz [파일명]</code>	파일 압축 xz의 경우 압축 파일이 훨씬 더 적어짐
<code>xz -d [파일명]</code>	압축 풀기

### -tar 이용법

<code>tar -cvf 이름.tar 추가 파일</code>	추가 파일은 오른쪽과 같이 나열	<code>tar -cvf abc.tar a.txt b.txt</code>
<code>tar -xvf 풀이름.tar -C ./폴더</code>	폴더는 미리 생성,(C 가 대문자 인것에 주의)	<code>tar -cvf abc.tar a.txt b.txt</code>

-앞에 j를 붙여 `jcvt jxvf`로 사용하면 `tar.xz` 사용법이 됨

- 가장 선호 되는 방식 (압축용량이 가장 작다.)

## 1. tar.xz로 압축하기

- 옵션 Jcvf

## 2. test1 에 압축파일 해제

- 옵션 Jxvf

## 3. tar.gz 으로 압축하기

- 옵션 zcvf

## 4. test2에 압축파일 해제

- 옵션 zxvf

## 5. tar.xz 와 tar.gz의 압축률 비교하기

### 파일 생성하기

---

echo 내용 > 이름.txt	txt가 생성되며 내용이 들어감
------------------	-------------------

---

echo 추가 내용 >> 이름.txt	기존 내용 밑으로 추가내용이 들어감
----------------------	---------------------

### 반복문(for)

---

```
for((i =0 ; i < 5 ; i++))do echo "#"; done    # 이 5번출력, 괄호가 두개인 점에 주의 하자
```