Make, ctags

Prof. Seokin Hong

Kyungpook National University

Fall 2019

Makefile

- Compiling the source code files can be tiring, especially when you have to include several source files

- **Makefile** is a special file that help build and manage the projects automatically
  - Simplify building program executables that may need various modules
  - Determine how the modules need to be compiled or recompiled together

- For example, assume we have the following source files.
  - main.cp, hello.cp, factorial.cp, functions.h
  - $ gcc main.cpp hello.cpp factorial.cpp -o hello ????

    or
  - $ make

An example

```
CFLAGS = -Wall -g
CC = gcc
LIBS = -lm

hello: main.o factorial.o hello.o
	${CC} ${CFLAGS} -o hello main.o factorial.o hello.o ${LIBS}

main.o : main.c
	${CC} ${CFLAGS} -c main.c -o main.o

factorial.o : factorial.c
	${CC} ${CFLAGS} -c factorial.c -o factorial.o

hello.o : hello.c
	${CC} ${CFLAGS} -c hello.c -o hello.o

clean:
	rm -f factorial.o hello.o main.o hello
```

Rules for Makefile

- The general syntax of **Makefile** target rules

```
[target] : [dependent ....]
    [ command ...]
```

  - $ make target
    - **make** finds the target rule that applies

```
hello: main.o factorial.o hello.o
	$(CC) main.o factorial.o hello.o -o hello
main.o : main.c
    $(CC) -c main.c
factorial.o: factorial.c
    $(CC) -c factorial.c
hello.o: hello.c
   $(CC) -c hello.c
```

    - if any of the dependents are newer than the target, **make** executes the commands one at a time
    - If any dependents have to be made, that happens first

Dependency

- A final binary will be dependent on various source code and source header files.
- Dependencies specify source codes for any target binary.
  - Ex) hello is dependent on main.o, factorial.o, hello.o files. **Hence, whenever there is a change in any of these object files, make will take action**

```
hello: main.o factorial.o hello.o
	$(CC) main.o factorial.o hello.o -o hello
```

  - **How to** prepare .o files

```
main.o : main.c
    $(CC) -c main.c
factorial.o: factorial.c
    $(CC) -c factorial.c
hello.o: hello.c
    $(CC) -c hello.c
```

Macros

- Similar to variables.
  - Macros are defined in a Makefile as = pairs

```
CFLAGS = -O -systype bsd43
LIBS = "-lcurses -lm -lsdl"
CC = gcc
```

- Special Macros
  - $@ : name of the file to be made
  - $? : names of the changed dependents

```
hello: main.c hello.c factorial.c
    $(CC) $(CFLAGS) $? $(LDFLAGS) -o $@

Alternatively:
hello: main.c hello.c factorial.c
    $(CC) $(CFLAGS) $@.c $(LDFLAGS) -o $@
```

$@ represents *hello*
$? or $@.c picks up all the changed source files

Macros

- Special Macros (Cont'd)
  - $< : name of the related file that caused the action
  - $* : prefix shared by target and dependent files.

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

Alternatively:

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

Rules for Makefile

- Common targets in Makefiles
  - **Make all** : compiles everything so that you can do local testing before installing applications
  - **make install** − installs applications at right places.
  - **make clean** − cleans applications, gets rid of the executables, any temporary files, object files, etc.

```
clean:
    rm *.o
```

An example

```
SRCS = main.c factorial.c hello.c
OBJS = $(SRCS:.c=.o)
TARGET = hello
CFLAG = -Wall -g
LFLAGS = -L/home/test/lib -L../lib
INCLUDES = -I../include
LIBS = -lm
CC = gcc

all: ${TARGET}

hello: ${OBJS}
	${CC} ${CFLAGS} -o $@ ${OBJS} ${LFLAGS} ${LIBS}

.c.o:
	${CC} ${CFLAGS} ${INCLUDES} -c $<

clean:
	rm -f ${OBJS} ${TARGET}
```

ctags

- Makes it easy to navigate large source code projects
- Provides some of the features that you may be used to using in Eclipse or other IDEs
  - jump from the current source file to definitions of functions
  - jump structures in other files

- Install ctags
  - $sudo apt-get install ctags
- Run ctags to generate the tags file
  - $ctags –R
- Generate the tag file in vim
  - :!ctags -R

Using ctags from vim

- vim will automatically pick up your tags file
- ctrl +] : jump to the definition of a function or a structure
- ctrl + t : go back to the previous location
- :ta function_name → go directly to a tag's definition
- :ts → shows the list
- :tn → go to the next tag in that list
- :tp → go to the previous tag in that list
- :tf → go to the first tag of the list
- :tl : goes to the last tag of the list
- :tags → show the tags you've traversed since opened vim

Navigating Linux Kernel with Ctags

$ git clone https://github.com/torvalds/linux.git

$cd linux

$ctags -R