GNU make

(Operating System)

Dr. Mayank Pandey

Motilal Nehru National Institute of Technology Allahabad, *Prayagraj*

Consider The Scenario

- Building a project or package involving a number of source files.

- Example : files *hellomake.c*, *hellofunc.c* and *hellomake.h* need to be compiled together into a binary called *hellomake*

- Traditional approach : use the compiler
  - *gcc -o hellomake hellomake.c hellofunc.c -I*

Problems

- Traditional approach suffers from some problems :
  - If this has to be done on multiple systems, the same set of commands has to be typed repeatedly.
  - Even if one of the files are updated, all the files are going to be recompiled, which is inefficient.
  - Practical projects might contain hundreds of files, which just amplifies the inefficiencies.

~ There should be some way to automate this process.

make

- Make allows us to automate the process of building target file(s) from source file(s).
- At the basic level, it can be used to perform simple compilation of multiple program files.
- At the advance level, it can allow us to cascade changes in a single file to multiple dependant files and send the details to the concerned users as an email.
- Or any arbitrary source to target development.

Make (continued)

- Uses timestamps to avoid processing unnecessary files.
- If the timestamp of target is newer than all the sources, do nothing.
- Else, one of the source files have been updated after the target, therefore the target will be rebuilt.

Makefile

- A file that can be **only** named makefile, Makefile or GNUmakefile
- It is the “program” that make uses to build the target.
- Contains rules indicating the targets that are to be built using which source files using which compilers along with related arguments passed.
- Target can be a file or an action to be taken, in the latter case it is called a “phony target”.

Makefile (continued)

- Rules are of the form :
  - **target** : **source_1** <space> **source_2** <space> **source_n**
  - <tab> **command**
  - **source_1** : **sub_source_1** <space> **sub_source_n**
  - <tab> **command**
  - ...
  - ...
  - so on and so forth.

- The "**command**" defines what has to be actually done to the source files to produce the target. It can also contain shell keywords like touch, echo, rm etc.

Example: 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
```

Variables

- In the previous examples some words pop up repeatedly, example “cc”, “-o” etc.
- To allow cleaner code and ease of development, make allows using variables called macros.
- Example :
  - CC = gcc
  - CFLAGS = -o

- These can later be accessed as : $(CC) and
- $(CFLAGS)

Example

```
objects = main.o kbd.o command.o display.o \
          insert.o search.o files.o utils.o

edit : $(objects)
        cc -o edit $(objects)
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 $(objects)
```

Exercise 1

- Try to add macros in the previous example as much as possible and run the new makefile.

Conditionals

- Ifeq(if equal), ifneq(if not equal) etc.
- Syntax :
  - ifeq ($(var_nam),”value”)
  - text-if-true
  - else
  - text-if-false
  - endif
- For example if file_type = java use java compiler, else use C compiler.