Makefile

A multi-file dependency resolver

Presented By:
Nilabja Sarkar
Kashish Gupta
Shaik Zunaid Ahamed
Under the guidance of **Dr. Anjeneya Swami Kare**

Why?

- Small program can store and run in a single file
- If not a small program we need multiple files

  > many line of code

  > multiple components

Problems

- Large file harder to manage
- Every change required long compilation
- Multiple files can’t modify simultaneously

continue..

Solution :

- Divide project in multiple files
- Need good division component
- Minimum compilation if changes
- Easy maintenance of project structure
- Can use DAG (directed acyclic graph) structure

Makefile syntax

Target : dependencies

    actions

- Target - a file you want to generate
- Prerequisite - files required to generate the target file
- Command - it is used to generate target

A Makefile must be named “Makefile” without any extensions and to run type “make” in terminal.

HERE, don’t use spacebar for indentation use tab as it gives error.

used as comments

Makefile ALL

A Makefile will only try to generate the first target listed. Use “all” to generate multiple files or only compile list of standard targets.

SYNTAX :

all : target

At the time of running use “make all” command which results in compiling targets.

Makefile Clean

After compiling several *.c and *.h files using make . It deletes up the executable file, all the object files, or some other files from directory.

SYNTAX :

clean : -rm *.o filename

To invoke it, type “make clean” on terminal.

Project maintenance

- Done by UNIX make command
- The make command reads a makefile understands the project structure and makes it executable
- Project structures and dependencies can be represented as DAG

EXAMPLE

- Project contains 3 files
- main .c sum.c sum.h, where sum.h included in both .c files
- Executable should the file sum

Makefile

```
sum: main.o sum.o
  gcc –o sum main.o sum.o
main.o: main.c sum.h
  gcc –c main.c
sum.o: sum.c sum.h
  gcc –c sum.c
```

Equivalent makefile

- .o depends (by default) on corresponding .c file. Therefore, equivalent makefile is:

```
sum: main.o sum.o
  gcc –o sum main.o sum.o
main.o: sum.h
  gcc –c main.c
sum.o: sum.h
  gcc –c sum.c
```

continue..

•We can compress identical dependencies and use built-in macros to get another (shorter) equivalent makefile :

```
sum: main.o sum.o
  gcc –o $@ main.o sum.o
main.o sum.o: sum.h
  gcc –c $*.c
```

Make operation

- Project dependencies tree is constructed
- Target of first rule should be created
- We go down the tree to see if there is a target that should be recreated. This is required when the target file is older than one of its dependencies
- In this case we recreate the target file according to the action specified, on our way up the tree. Consequently, more files may need to be recreated
- If something was changed, linking is performed

continued...

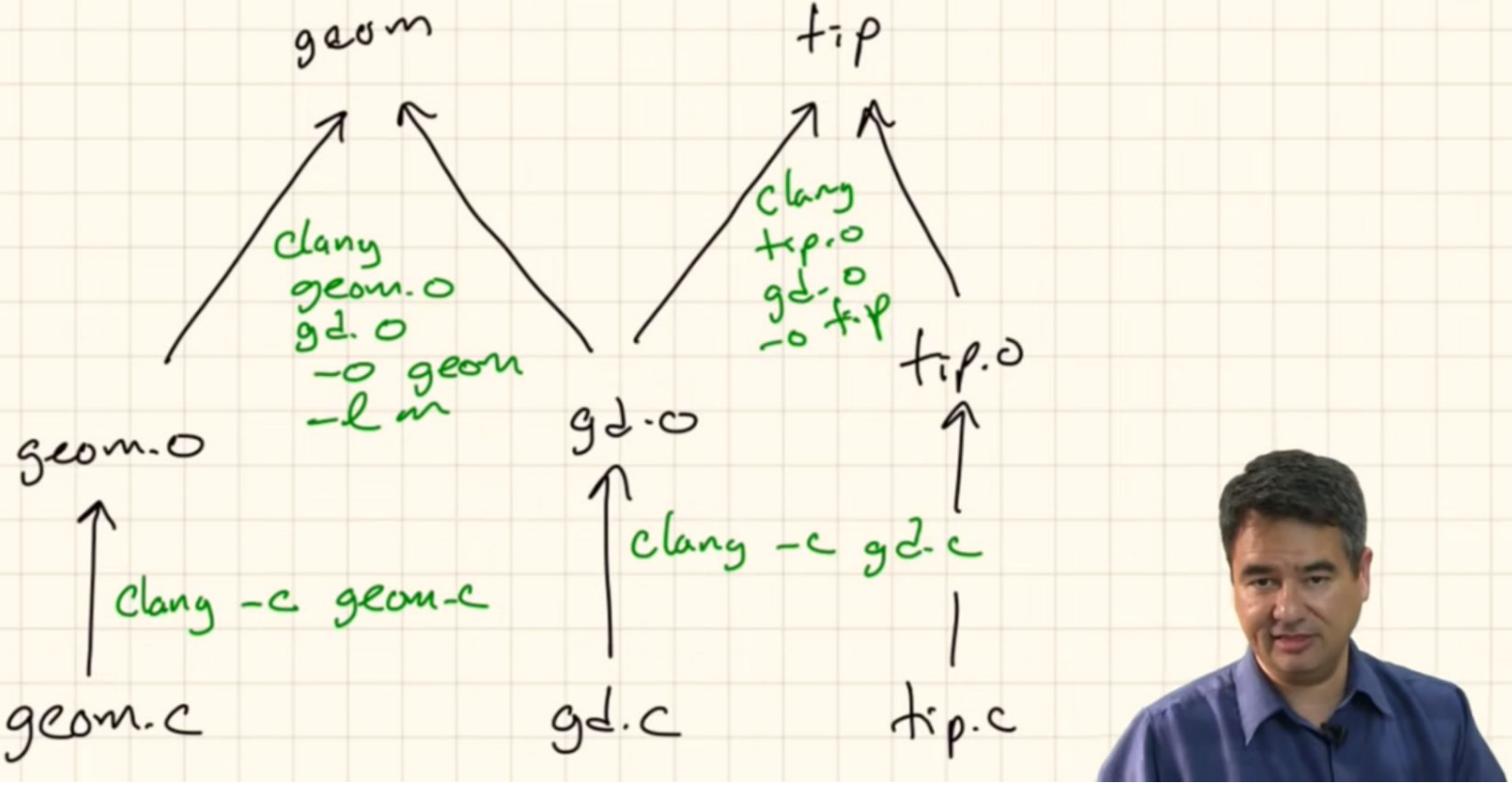
- make operation ensures minimum compilation, when the project structure is written properly

- Do not write something like:

  prog: main.c sum1.c sum2.c

  gcc –o prog main.c sum1.c sum2.c

  which requires compilation of all project when something is changed


geom
tip
clang
geom.o
gd.o
-o geom
-lm
clang
tip.o
gd.o
-o tip
tip.o
geom.o
gd.o
clang -c geom.c
clang -c gd.c
geom.c
gd.c
tip.c

continued..

gcc −c main.c

gcc −o sum main.o sum.o

- main.o should be recompiled (main.c is newer).
- Consequently, main.o is newer than sum and therefore sum should be recreated (by re-linking).

Makefile example (another)

```
#to compile
#------- method 1--------
#final:
#	gcc main.c hello.c sum.c -o final

####make
#./final
#--------- method 2 -----

$(CC) = gcc
final:
		$(CC) main.c sum.c hello.c -o final
clean:
		rm *.o final
##make final
```

continue

```
#-------method 3-------
#$(CC)  = gcc
#final : main.c sum.c hello.c
#	$(CC) -c main.c
#main.o: main.c header.h
#$(CC) -c hello.c
#sum.o: sum.c header.h
#	$(CC) -c sum.c
```

continued..

- We can define multiple targets in a makefile
- Target clean – has an empty set of dependencies. Used to clean intermediate files.
- make

–Will create the compare_sorts executable

- make clean

–Will remove intermediate files

Reference

https://edoras.sdsu.edu/doc/make.html

https://www.dartmouth.edu/~rc/classes/soft_dev/make.html

https://www.youtube.com/watch?v=GExnnTaBELk&t=115s

Thank You