Making makefiles

Tutorial for CS-2810

Subhashini V

IIT Madras

Outline

Introduction
- Motivation

Compiling
- Object file

Makefiles
- Macros
- Naming
- Phony-Targets

Example

Extras
- Default-rules
- Remarks

The why, what and how of makefiles for you and me.

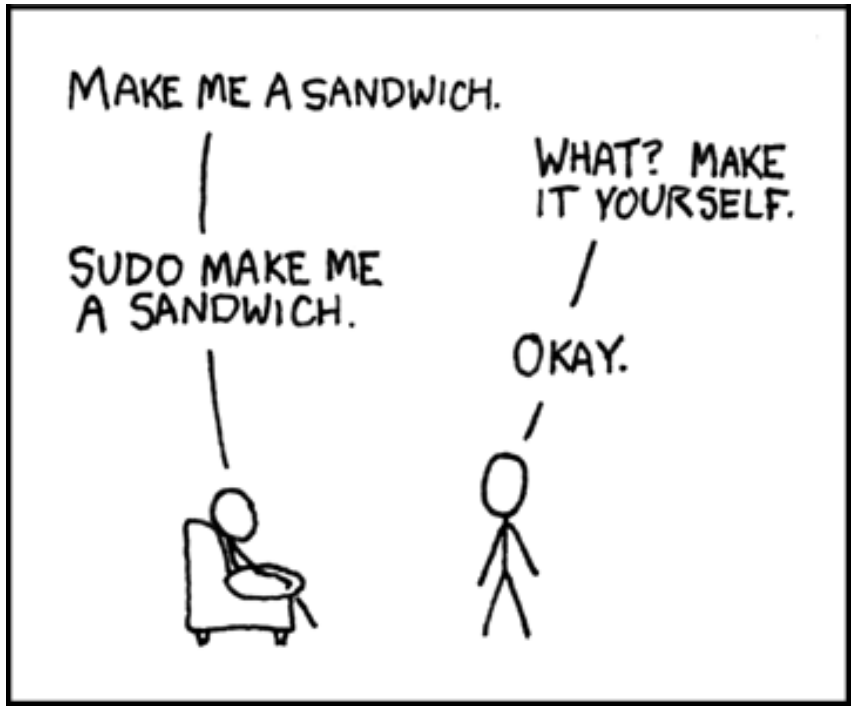

Figure: xkcd

(comic is for sudo, but who cares, it has 'make')

Why makefiles?

- Lots of source files: foo1.h, foo2.h, foobar1.cpp, foobar2.cpp
- How to manage them all?
- Compiling is complicated!!!

Why makefiles?

- Lots of source files: foo1.h, foo2.h, foobar1.cpp, foobar2.cpp
- How to manage them all?
- Compiling is complicated!!!
- Solution: make

What are makefiles?

- make - automagically build and manage your programs.
- compile quickly with a single command
- recompile is even quicker

How does it work?

- Give targets (usually a file to be created)
- Specify dependencies for each target.
- Give command to create target from dependencies.
- '**make**' recursively builds the targets from the set of dependencies.
- recompilation - time-stamp of modification

.h and .cpp

.h files contain

- declarations.
- functions that the class promises.

.cpp files contain

- definitions
- implementations of the promised methods.
- other functions that are not a part of any class.

BigInt assignment example

The BigInt data type in RSA

- BigInt.h - with proposed methods : +, -, *, /, !
- BigInt.cpp - with implementation of methods
- rsa.cpp - ONLY needs #include “BigInt.h”.
  Don’t need BigInt.cpp.

Creating Object file

object file or (.o file)

- For creating rsa.o, no need to include BigInt.cpp!
- Eventually need BigInt.cpp for running.
- Link later.
- You also don't need **main()**

Command:

```
g++ -Wall -c rsa.cpp
```

* -Wall : all warnings turned on
* -c : Compiles but doesn't link

Makefiles!

A makefile describes the dependencies between files.

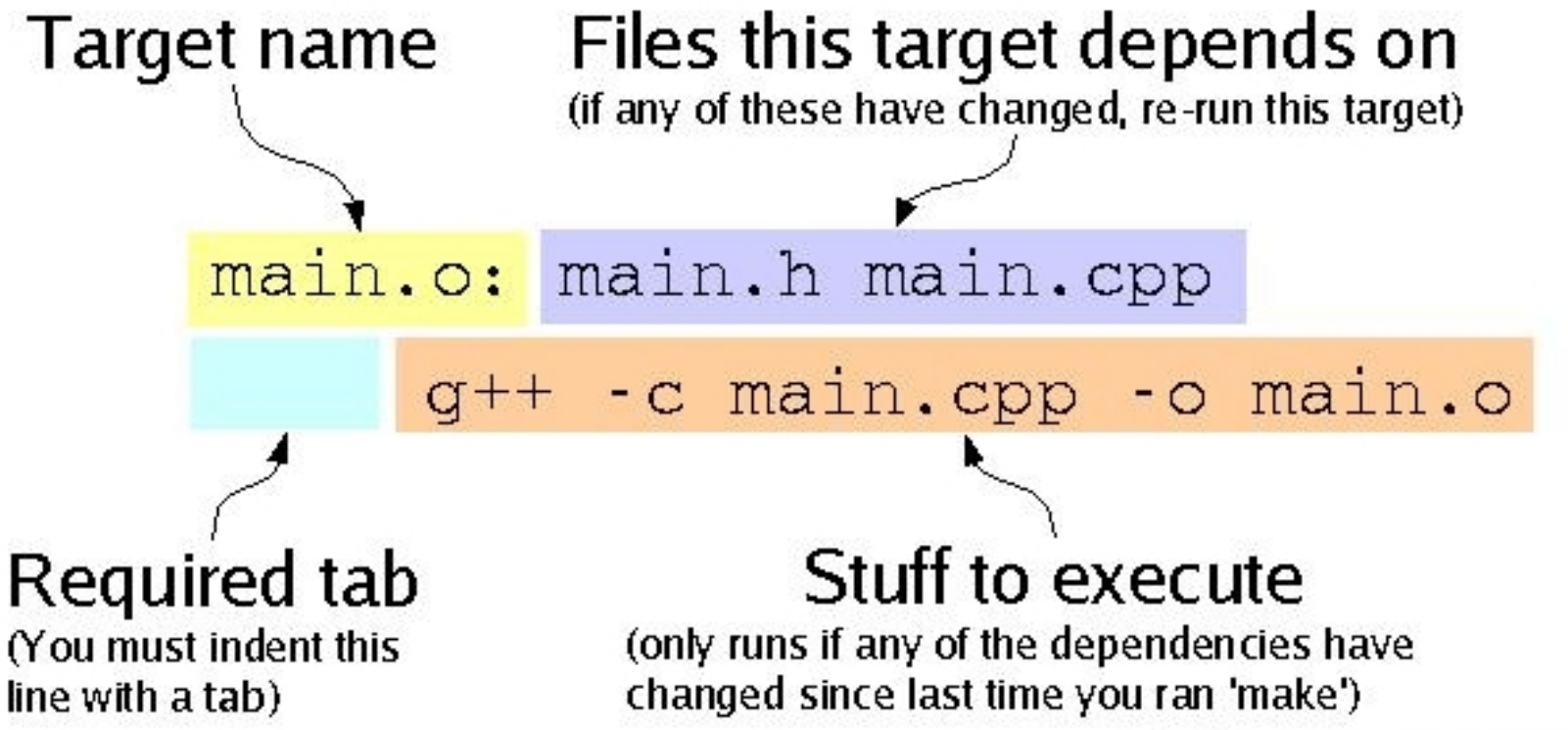

Figure: Content of a Makefile

Targets, Dependencies, Commands

- Target is usually a file. Remember ":"
- Dependencies : .cpp, corresponding .h and other .h files
- Determining dependencies.
- the all important *[tab]*
- Commands : can be multiple

Use macros!

Like #define

```
OBJS = rsa.o main.o
CPP = g++
DEBUG = -g
CPPFLAGS = -Wall -c $(DEBUG)
LDFLAGS = -lm
```

Usage: $(variable)

```
rsa :  $(OBJS)
        $(CPP) $(LDFLAGS) $(OBJS) -o rsa
```

- Automatic variables : $@ , $<, $^ ...

Naming the file

Naming and running the make file.

- **makefile** - run with **make**
- **Makefile** - run with **make**
- give any *filename* - run as **make -f** $<$ *filename* $>$

make picks the first *target* in the file and executes the *commands* if the *dependencies* are more recent than the *target*.

Phony targets

Dummy targets to run commands. They don't actually create files.

- make all
- make clean
- make tar

Use .PHONY or directly the name.

```
.PHONY: clean
clean:
	\rm -f *.o *~ rsa
```

Or

```
clean:
	\rm -f *.o *~ rsa
```

Makefile

```
OBJS = rsa.o main.o BigInt.o
CPP = g++
DEBUG = -g
CPPFLAGS = -Wall -c $(DEBUG)
LDFLAGS = -lm

rsa :  $(OBJS)
        $(CPP) $(LDFLAGS) $(OBJS) -o rsa
main.o :  main.cpp BigInt.h rsa.h
        $(CPP) $(CPPFLAGS) main.cpp -o $@
rsa.o :  rsa.cpp rsa.h BigInt.h
        $(CPP) $(CPPFLAGS) rsa.cpp -o $@
BigInt.o :  BigInt.cpp BigInt.h
        $(CPP) $(CPPFLAGS) BigInt.cpp -o $@
clean :
        rm -f  *.o  *~ rsa
```

Automatic Variables

`$@ , $<, $^`

- `$@` : used for the target variable.
- `$<` : the 1st prerequisite.
- `$^` : is like “all the above” - all the prerequisites.

Default automatic rules to compile

Example-1

```
prog:  foo.o foobar.o ...
<TAB> (nothing)
```

```
$(CPP) $(LDFLAGS) $^ -o $@
```

(**if** prog.o (or prog.c) is one of the prerequisites.)

Example-2

```
prog:  prog.c prog.h
<TAB> (nothing)
```

```
$(CPP) $(CPPFLAGS) -c $< -o $@
```

Check these.

‘make’ from within the editor

Some editors like Vim allow you to call ‘make’ while still editing the makefile :)

- **:make** - to run the Makefile.

To navigate through the errors and fix compilation issues fast, try

- **:copen** - Opens the quickfix window, to show the results of the compilation.
- **:cnext** or **:cn** - jump to the next error in source code.
- **:cprev** or **:cp** - jump to the previous error in source code.

Remarks

- Don't forget to put the **TAB** before entering the command.
- Look at examples of Makefiles.
- Lots of tutorials and resources available online.

Questions?

Just **make** *it.*