# Lesson 2 - Libs

# Libs

Some times we need to devide our code for separate files. Moreover , we need to devide it for reusable parts, and client specific.

For this, the Library was envented.
A stand-alone, already compiled (and probably tested) bunch of code, for developers to be used.

# Libs

There is at least two Libraries types:
Static and Dynamic (also called Shared)

Static usualy ends with “.a” .
In compile time, they are integrated in the executable file.

Dynamic ends with “.so” (shared object).
In compile time, they are linked to the code, but not included in executable, and can be replaced

# Libs - shared

Name convention:
Lib name will be lib<name>.so
f.ex libCore.so, libAlgoImageTracking.so

Compilation of the lib:
gcc -o libHello.so -shared -fPIC hello_ariel.c

# Libs - shared

Compilation of code:
gcc main1_1.c -L. -l Hello -o hello2

-L measn where to look for the library

-l (small L) measn whats the name of the library.
note that the “lib” and “.so” are omited

export LD_LIBRARY_PATH=.
updates LD, where to look for the lib in run time

# Makefile in 5 minutes

Propably should include an “all” task, and “clean”

The structure is:
task name : dependency (files or tasks)
(tab) cmd command (gcc...)

.PHONY: clean “header” will become before a command, independed of the filesystem (otherwise,if you will have file named “clean” it will make a bug)

# Libs - tools

How can we realy know what is in our library and what it made for ?

*file* – shows some basic info. Is it executable or shared lib, for that platform (x86,arm) was build

*nm* – shows that symbols are inside. Our functions shoud be listed too. (note that c++ will change funtions name)

# Libs - executable

How can we know that our executable depends on libraries ?

*ldd* – will display the dependencies of an executable.

*ltrace* – will display library calls

*strace* – will display system calls

# Libs – dynamic loading

Can we load a library without compiling against it?

dlopen – load the library (lasy – when needed)

dlsym – looks for a symbol (function name)

Example !