**Alexander Kläser**

Home

Research

Software

Publications

Contact

*Back to:*

*LEAR - Home*

*LEAR - Team*

# Generic Makefile with automatically created link dependencies in C/C++

It has been a while that I was looking for a way how to create and update **link** dependencies automatically from a .cpp/.c file. The idea is: you only tell in the Makefile "this source file is an exectuable" and make finds all files that need to be linked with. I put the solution that I developed here on my site hoping that it may help others, as well. Any comments/questions are welcome, just drop me a line.

***Acknowledgments:***
*(July 2009) Thanks to Dan Sanduleac for pointing out a bug that he came along in dep.py.*

**Features:**

- Automatic generation of **link dependencies** (i.e., all files an executable need to be linked with are found automatically).
- Automatic generation of header dependecies (i.e., if an included header file changes, depending source files are recompiled).
- Everything is done using make and one python script, no extra overhead.
- Temporary files (.o files and files created by the generic Makefile) are stored in a separate directory.

**Rules to obey:**

- A source file 'MySuperbObject.cpp' or 'shared/cool_functions.c' needs to have a corresponding header file with the same name, i.e., 'MySuperbObject.h' and 'shared/cool_functions.h'.
- You need to use absolute include paths or paths below the directory where your Makefile is stored. For instace for gcc only use the option '-I' with paths like these '-I /include/path' or '-I sub/include/path'. (Temporary files are put into a build directory, relative paths including '..' might step outside of this build directory.)

**How to use it?** The use is easy. Download the dependency python script and download the generic include Makefile. Adjust in the downloaded generic Makefile (generic.mk) the path ('DEP=...') to the dependency python script (dep.py) you downloaded. Create your own Makefile in which you include the generic Makefile ('include /.../generic.mk'), define a list of target executables ('TARGETS=...') and you are done. Below you find two examples of how a Makefile may look like.

**How does it work?** You set in the Makefile the *target* files, i.e., files that should be compiled and linked as executable files. The Makefile generates *dependency* files (.d files) for each target file, i.e., dependencies on all files that are included recursively through the target file and through the files it includes (directly and indirectly). Dependency files are generated using gcc with the -MM option. A python script (dep.py) then parses the dependy files and it checks whether there exist header files with a corresponding .c or .cpp file. And this is the assumption for this to work: if a header file is included and there exists a corresponding source file (.c or .cpp) in the same directory, we need to compile the source file and link it to our target exectuable. These files are thus added to the list of files we need to link against in the corresponding *link dependency* file (.l files). Consequently, dependency files are generated for these source files, as well. These new dependency files are investigated for new source files that we need to link against. And this continues recursively until all files are found that need to be compiled and linked to our target executable. Actually a seperate directory (by default '.build') is created and all created files (.d, .l) and all object files (.o) are stored in this directory. The final executables are stored by default into the directory 'bin'.

## Simplistic Makefile

Here is a simple example of a Makefile assuming you have a file 'my_program.cpp' in the same directory as the Makefile:

```
# set the target executables that need to be built
TARGETS := my_program

# set some flags and compiler/linker specific commands
CXXFLAGS = -ggdb -O2 -DNDEBUG
LDFLAGS = -Wall -ggdb
include /path_to_generic_makefile/generic.mk
```

Then you can type 'make all' to compile and link all executables or 'make clean' to clean up.

## Advanced Makefile with extra features

And here a more advanced Makefile that I tend to work with. Note that it has some extra features you might or might not be interested in. However, all you need to do is illustrated in the simplistic Makefile above. This Makefile only shows some other features that might be of interest to others.

It assumes that the source files (.cpp files) for the executables are in the directory 'tools'. The default variables for build and bin directory (BUILDDIR, BINDIR, documented in generic.mk) are overridden with user-defined names. Support for 32bit and 64bit compilation is included. I defined a standard set of libraries and a standard compiler (g++). For the executable 'myComplicatedTool', I use a particular set of libraries and a particular compiler (here mpiCC for MPI).

```
# set the binaries that have to be built
SRCS := $(wildcard tools/*.cpp)
TARGETS := $(TARGETS) $(SRCS:.cpp=)

# set the build configuration set
BUILD := release
#BUILD := debug
BIT := 64
#BIT := 32

# set bin and build dirs
BUILDDIR := .build_$(BUILD)$(BIT)
BINDIR := $(BUILD)$(BIT)

# include directories
INCLUDEDIRS := \
        common_code \
        $(HOME)/more_common_code

# library directories
LIBDIRS :=

# set which libraries are used by which executable
LDLIBS = $(addprefix -l, $(LIBS) $(LIBS_$(notdir $*)))
LIBS = cv boost_program_options-mt-d boost_serialization-mt-d boost_regex-mt-d
LIBS_mySimpleTool :=
LIBS_myComplicatedTool := highgui avformat avcodec avutil

# set some flags and compiler/linker specific commands
CXXFLAGS = -fopenmp -m$(BIT) -pipe -D STD=std -Wall $(CXXFLAGS_$(BUILD)) $(addprefix -I, $(INCLUDEDIRS))
CXXFLAGS_debug := -ggdb
CXXFLAGS_release := -ggdb -O3 -DNDEBUG
LDFLAGS = -m$(BIT) -pipe -Wall $(LDFLAGS_$(BUILD)) $(addprefix -L, $(LIBDIRS))
LDFLAGS_debug := -ggdb
LDFLAGS_release := -ggdb -O3

# we set specific compilers for specific tools
CXX = $(if $(CXX_$(notdir $*)), $(CXX_$(notdir $*)), g++)
CXX_myComplicatedTool := mpiCC

include /path_to_generic_makefile/generic.mk
```

by Alexander Kläser 2010