Linux Users Group

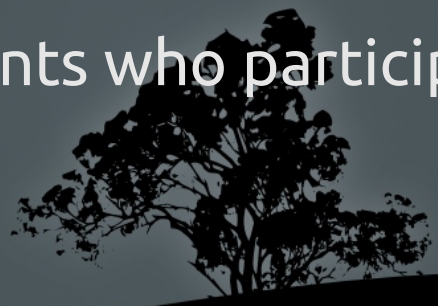
UT Arlington

How to compile C/C++ programs on Linux with the GNU toolchain

Rohit Rawat

About LUG@UTA

- The Linux user group is an officially recognized student organization at UTA.
- The goals of the Linux user group are to:
  - promote the use of the Linux operating system and other open source software.
  - provide a venue for students to gain experience giving technical presentations.
  - provide a forum for students to learn about new aspects of Linux and open source software.
  - promote fellowship among students who participate in the group.



Why join?

- Expand your knowledge – past topics include using LaTex, using GPG, geeky stuff
- Meet new people
- Build leadership skills – we have many unclaimed officer positions right now

More information

- LUGUTA@LISTSERV.UTA.EDU
- http://luguta.org/
- MavOrgs – search for LUGUTA

GNU toolchain

- **GNU Compiler Collection**
  - **Suite of compilers – C, C++, Java**
- GNU make
  - Utility to automatically execute build steps
- GNU Binutils
  - Tools to create binaries
- GNU Debugger
- GNU autotools
  - Make portable packages

GCC C/C++ front-end

- gcc inputfile.c
- g++ inputfile.cpp
- Default executable name 'a.out'
- To run:

  ./a.out

Common gcc options

- -o outputfile (give an alternate name for a.out)
- -c (compiles only, doesn't link)
- -o3 (optimization level)
- -g (add debugging information)
- -Wall (displays all warnings)

Common gcc options

- -I includepath (additional locations to look for include files)
- -L librarypath (additional locations to search for libraries)
- -l libname (link against that library, e.g. -lm -lboost_system)

- Over 9000 options!

Editors (non-IDE)

- Emacs
- Vi
- Nano
- Gedit

nano

- Simplest console based editor
- Launched as:
  - nano filename
- Ctr-O to save the file.
- Ctr-X to exit.

GCC Java front-end

- gcj ClassName.java
- Default output: ClassName.class
- To run:

  java ClassName

GNU toolchain

- GNU Compiler Collection
  - Suite of compilers – C, C++, Java
- ***GNU make***
  - ***Utility to automatically execute build steps***
- GNU Binutils
  - Tools to create binaries
- GNU Debugger
- GNU autotools
  - Make portable packages

GNU make

- A project with mutiple source files:

```
g++ file1.cpp file2.cpp -o result
```

- Can get unmanageable for a bigger project.
- 'make' utility – looks for a file 'Makefile' for build instructions
- Automatically detects changed files and rebuilds only those files

Makefile

- Format:

```
target: prerequisites
[tab] commands
```

The tab on the second line is mandatory

- Default target is the first one in Makefile
- You can always specify one by name

  make counter

  - Will search for and build the target 'counter'
- 'clean' is a commonly used target to delete build results and temporaries
- 'install', 'check' are also commonly used target names

Simple makefile

```
hello: helloworld.cpp
        g++ -o hello helloworld.cpp
```

- Target file is 'hello'
- Depends on 'helloworld.cpp'
- The command to produce the target is:

```
        g++ -o hello helloworld.cpp
```

Implicit Makefile rules

CC = g++ *the default compiler to use*

CFLAGS = -I/include/dir *compiler flags*

LDFLAGS = -lm *linker flags*

$(CC) – using value of CC

%.c - wild card that matches all .c files

$@ - target name

$^ - list of dependencies

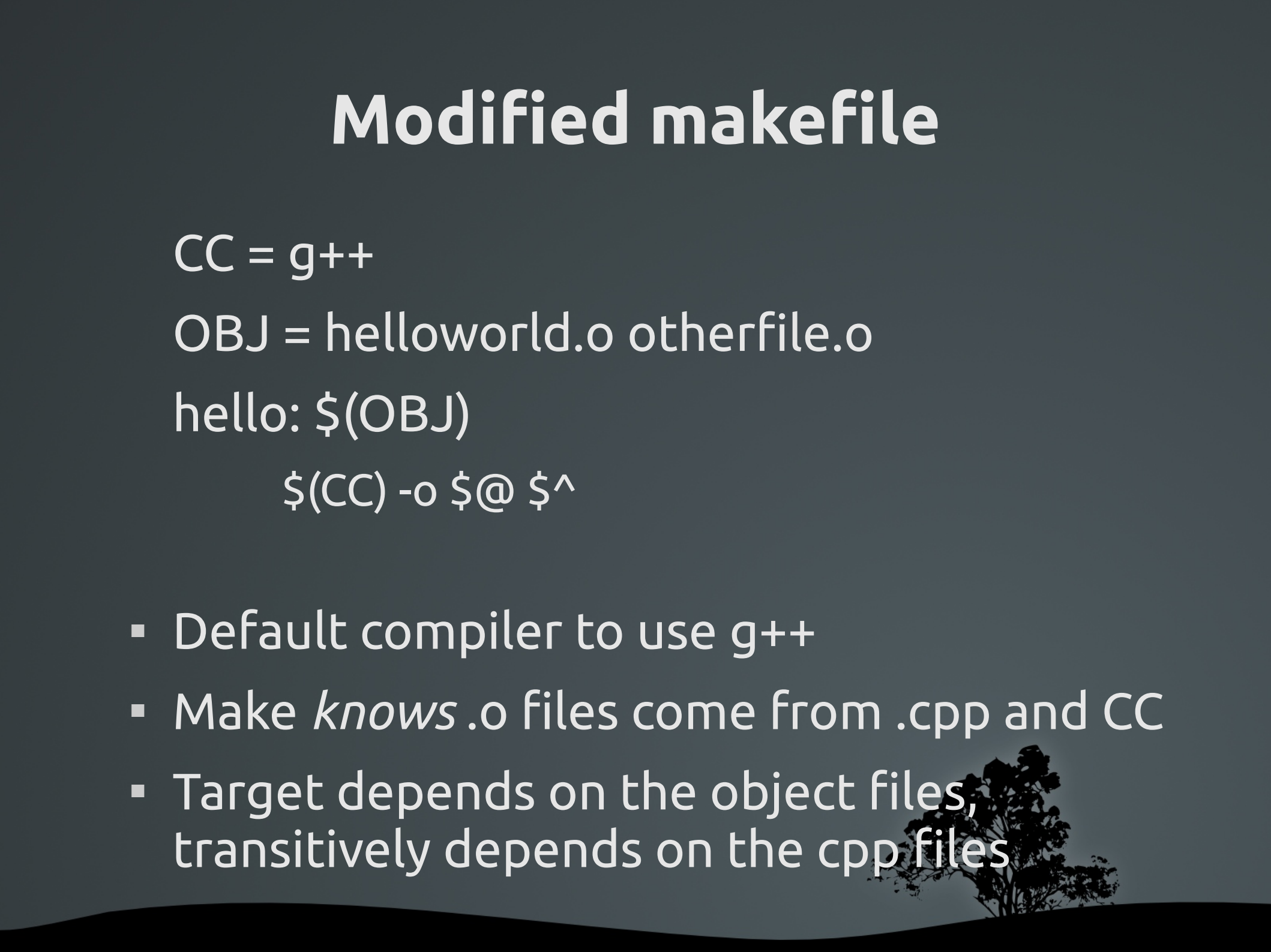
$< - first dependency in the list

.PHONY – defines targets with no output file

Modified makefile

```
CC = g++
OBJ = helloworld.o otherfile.o
hello: $(OBJ)
	$(CC) -o $@ $^
```

- Default compiler to use g++
- Make *knows* .o files come from .cpp and CC
- Target depends on the object files, transitively depends on the cpp files



Advanced Makefile

```
CC=gcc
CFLAGS=-I. -g
LDFLAGS=-lm
DEPS=hellomake.h
OBJ=hellomake.o hellofunc.o

default: hellomake

%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

hellomake: $(OBJ)
	$(CC) -o $@ $^ $(CFLAGS) $(LDFLAGS)

.PHONY: clean
clean:
	rm -f *.o *~ hellomake
```

- compiler flags
- linker flags
- depends on some .h files too

default target name

explicitly specify how each .o file will be built
making it dependent on the .h files

```
hellomake: $(OBJ)
	$(CC) -o $@ $^ $(CFLAGS)
```

specifies that clean is not a file target
this target deletes all the .o, temps, and the binary

GNU toolchain

- GNU Compiler Collection
  - Suite of compilers – C, C++, Java
- GNU make
  - Utility to automatically execute build step
- ***GNU Binutils***
  - ***Tools to create and manage binaries***
- GNU Debugger
- GNU autotools
  - Make portable packages

GNU binutils

- Some of the tools:
  - as – assembler
  - ld – the linker invoked by gcc/g++ to produce the executable
  - gprof – profiler/timing
  - ar – creates archives
  - objdump – information/disassembly tool
- Example: source code disassembly
  - objdump -S hellomake

GNU toolchain

- GNU Compiler Collection
  - Suite of compilers – C, C++, Java
- GNU make
  - Utility to automatically execute build step
- GNU Binutils
  - Tools to create binaries
- ***GNU Debugger***
- GNU autotools
  - Make portable packages

GNU Debugger - gdb

- Enable debugging information with -g during compilation and linking
- Try not to use optimization flags -oN
- Usage:
  - gdb program_name
- Allows: program listing, execution, stepping, breakpoints etc.

Sample gdb session

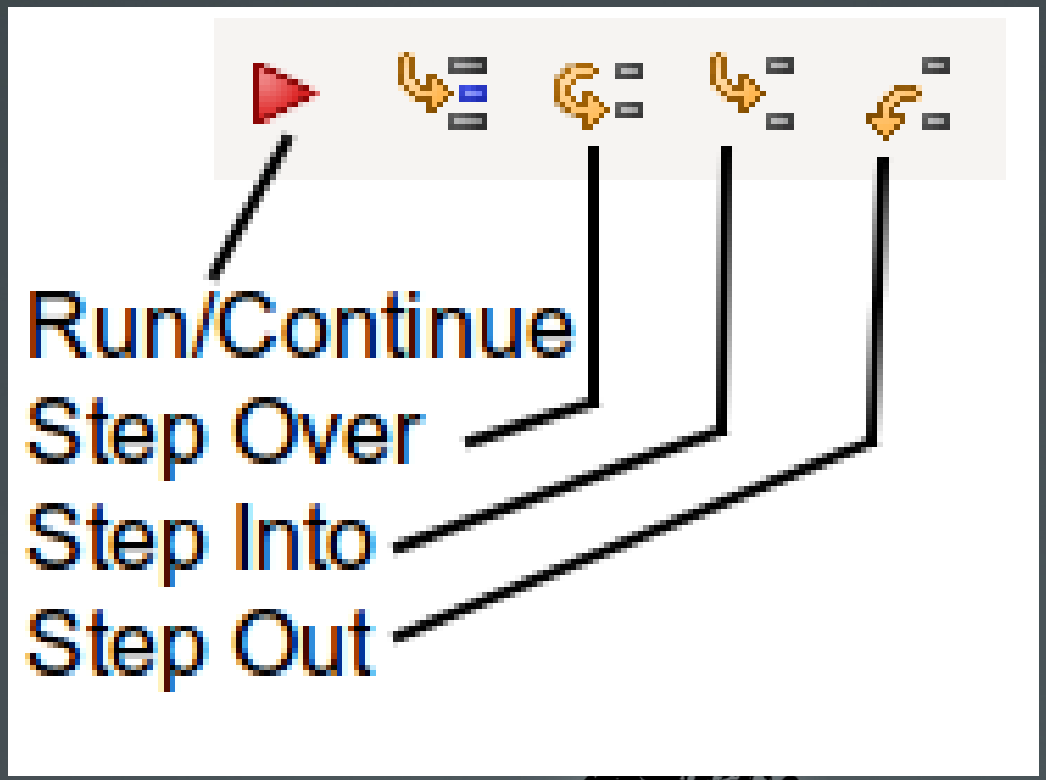
```
gdb hellomake
(gdb) list
1     #include <hellomake.h>
2
3     int main() {
4         // call a function in another file
5         myPrintHelloMake();
6         return(0);
7     }
8
(gdb) list myPrintHelloMake
1     #include <hellomake.h>
2     #include <stdio.h>
3
4     void myPrintHelloMake(void) {
5         printf("Hello makefiles!\n");
6         printf("Bye!\n");
7         return;
8     }
```

```
(gdb) run
Starting program: /home/rohit/LUG_Fall2013/prog/hellomake
warning: no loadable sections found in added symbol-file system-supplied DSO at 0x7ffff7ffa000
Hello makefiles!
Bye!
[Inferior 1 (process 13780) exited normally]
(gdb) break 6
Breakpoint 1 at 0x400540: file hellofunc.c, line 6.
(gdb) run
Starting program: /home/rohit/LUG_Fall2013/prog/hellomake
warning: no loadable sections found in added symbol-file system-supplied DSO at 0x7ffff7ffa000
Hello makefiles!
Breakpoint 1, myPrintHelloMake () at hellofunc.c:6
6         printf("Bye!\n");
(gdb) continue
Continuing.
Bye!
[Inferior 1 (process 13803) exited normally]
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000400540 in myPrintHelloMake
                                                   at hellofunc.c:6
        breakpoint already hit 1 time
(gdb) quit
```

Other execution control

- Press **Ctrl-C** to break immediately
- **next** (step over)
- **step** (step into)
- **finish** (step out)

- **print x**
- **set x = 10**



Watchpoints

- Break when a variable changes or is read
- Commands:
  - watch – break on write
  - rwatch – break on read
  - awatch – beak on read/write

Segmentation faults

```
(gdb) list
1     int func()
2     {
3             int *x;
4             x[99] = 100;
5             return 0;
6     }
7
8     int main()
9     {
10            func();
(gdb) run
Starting program: /home/rohit/LUG_Fall2013/prog/segfault
warning: no loadable sections found in added symbol-file system-supplied DSO at 0x7ffff7ffa00

Program received signal SIGSEGV, Segmentation fault.
0x0000000004004fa in func () at segfault.c:4
4             x[99] = 100;
(gdb) backtrace
#0  0x0000000004004fa in func () at segfault.c:4
#1  0x0000000000400515 in main () at segfault.c:10
```

GNU toolchain

- GNU Compiler Collection
  - Suite of compilers – C, C++, Java
- GNU make
  - Utility to automatically execute build step
- GNU Binutils
  - Tools to create binaries
- GNU Debugger
- **GNU autotools**
  - **Make portable packages**

GNU autotools

- automake & autoconf – automatically generates a system specific makefile and configure script
- libtool – build shared (dynamic) libraries portably
- gettext – localization for international languages

- Autotools provide a central framework for portable application development.

Common portability issues

- Many libraries and functions are not “standardized”
  - pow() may be defined in math.h or stdlib.h
- System/configuration specific symbols have to be defined in the makefile
  - gcc -DHAVE_OPENGL game.cpp
  - gcc -DHAVE_DIRECTX game.cpp
- Library locations differ on systems
  - /usr/bin/local/libs
  - /usr/local/

Possible solutions

- Use conditional compilation statements
- Write wrapper macros or functions

- Write shell scripts to guess setting and generate the makefiles (what is generally known as “./configure” step)

Auto library configuration

- pkg-config --list-all
- pkg-config --cflags opencv

  -I/usr/local/include/opencv
  -I/usr/local/include

- pkg-config --libs opencv

  /usr/local/lib/libopencv_calib3d.so
  /usr/local/lib/libopencv_contrib.so …

- Compilation command:

  g++ `pkg-config --cflags opencv` code.cpp `pkg-config --libs opencv`

References:

- GNU toolchain on Wikipedia: http://en.wikipedia.org/wiki/GNU_toolchain
- Nano editor: http://mintaka.sdsu.edu/reu/nano.html
- Makefiles: http://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html
- GDB: http://www.unknownroad.com/rtfm/gdbtut/gdbtoc.html
- GNU Autotools: http://www.lrde.epita.fr/~adl/dl/autotools.pdf

The vi editor

- More features but needs geting used to
- Launch: vi filename
- 'i' enters insert mode at cursor.
- 'Esc' key enters command mode, when in doubt hit it twice
- ':w' writes the file to disk
- ':q' quits, ':q!' quits without saving changes
- ':wq' saves then quits

Some commands I use

- '/' search
- 'a' insert after current character
- 'A' insert at end of line
- 'I' insert at beginning of line
- 'o' insert in a new line below
- 'dd'/'yy' delete or copy the current line
- 'p' paste
- 'cw' change word
- '#G' go to line number #