Makefiles

The Compilation Process

- Compiler stage
- Assembler stage
- Linker stage

```
g++ -o HelloWorld HelloWorld.cpp

./HelloWorld
```

Compile Commands

- Compiling multiple source files

```
g++ Bike.cpp Tricycle.cpp
```

- Compiling without linking

```
g++ -c Bike.cpp Tricycle.cpp
```

- Linking compiled code

```
g++ Bike.o Tricycle.o

g++ Bike.o Tricycle.o -o GoRide
```

Selective Compilation

```
g++ -c Bike.cpp
g++ Bike.o Tricycle.o main.o -o GoRide


g++ -c Tricycle.cpp main.cpp
g++ Bike.o Tricycle.o main.o -o GoRide
```

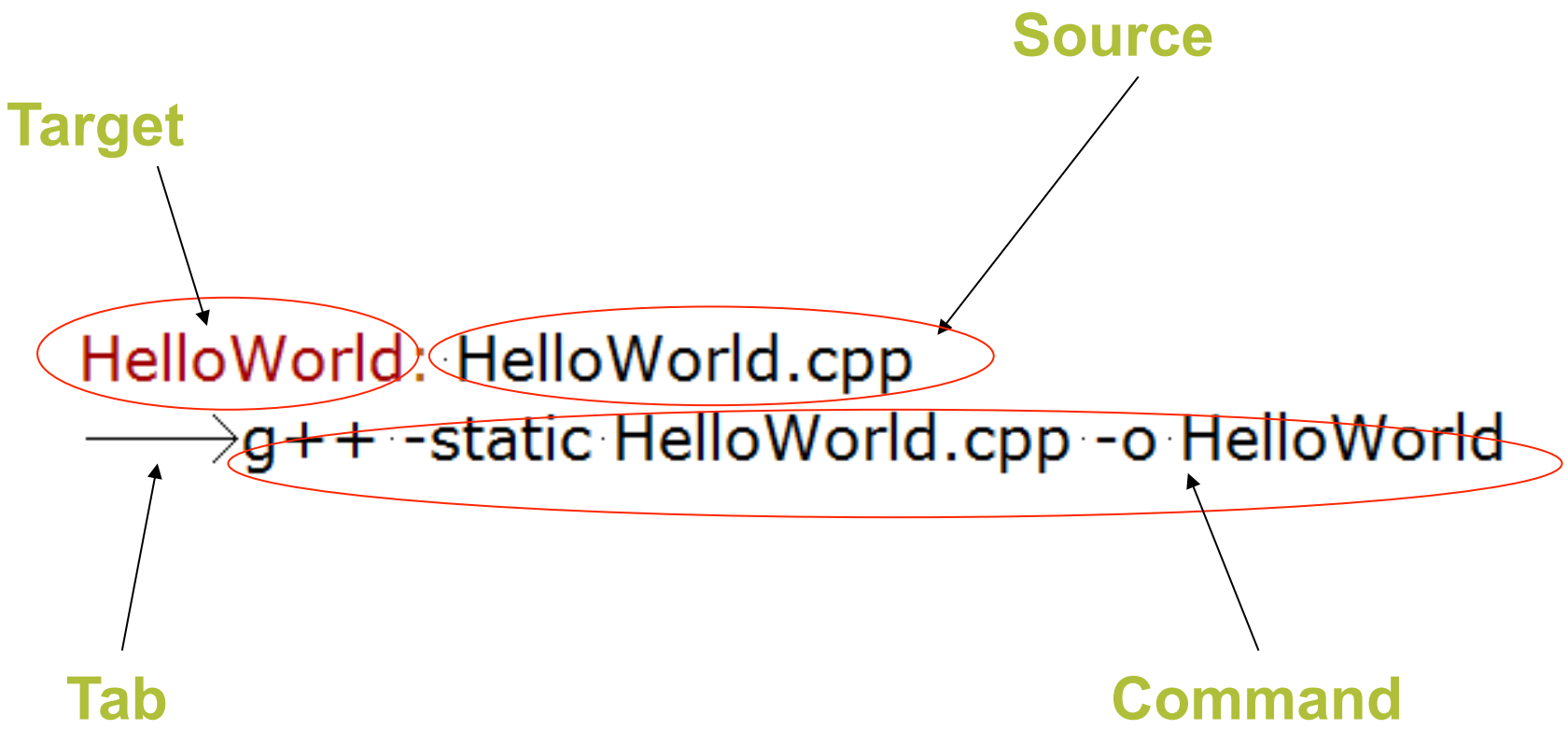
Compiler Flags

Flag	Usage
-o	To specify the output filename.
-w	Disable all warning messages.
-Wall	Enable most compiler warnings.
-Werror	Treat compiler warnings as errors.
-pedantic	Issue all the warnings demanded by ISO C++.
-pedantic-errors	Like `-pedantic`, except that errors are produced rather than warnings.
-static	On systems that support dynamic linking, this prevents dynamic linking with the shared libraries.

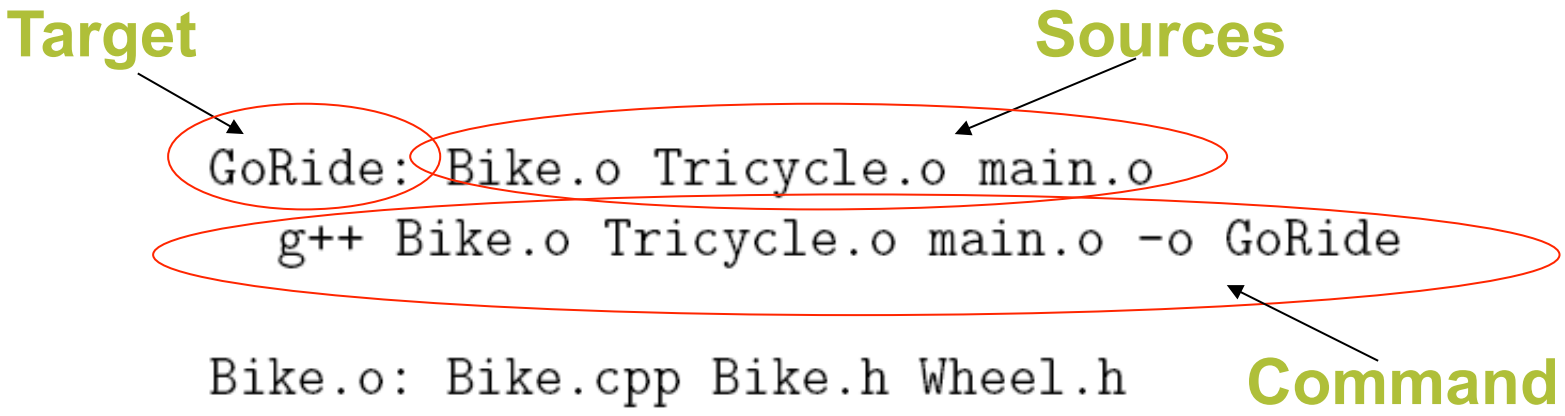
Compiler Flags

Flag	Usage
-o	specify the output filename
-Wall	turns on all compiler warnings
-g	turns on debugging. This makes your code ready to run under gdb.
-c	compiles it down to an object file, known as a '.o'. You can link together multiple object files into an executable. This is used in multiple file projects to reduce compile time.
-O	turns on optimization, you may also specify levels (-O2).
-E	outputs the preproccessor output to the screen (stdout).
-MM	outputs the Makefile dependancies for the cpp file(s) listed.

Sample makefile entry



Sample makefile




```
Bike.o: Bike.cpp Bike.h Wheel.h
   g++ -c Bike.cpp

Tricycle.o: Tricycle.cpp Tricycle.h Wheel.h
   g++ -c Tricyle.cpp

main.o: main.cpp Bike.h Tricycle.h
   g++ -c main.cpp
```

Order when linking

- .o files has to be listed in the correct order
- The make utility execute from left to right.
- Everything that a specific .o file are dependant on should be listed to the left of it.
  - All its parents
  - All classes of which it has instances

Custom command

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

```
make clean
```

Comments

```
# Linking the object code of the complete system:
GoRide: Bike.o Tricycle.o main.o
   g++ Bike.o Tricycle.o main.o -o GoRide

# Commands for partial compilation of c++ source files:
Bike.o: Bike.cpp Bike.h Wheel.h
   g++ -c Bike.cpp

Tricycle.o: Tricycle.cpp Tricycle.h Wheel.h
   g++ -c Tricyle.cpp

main.o: main.cpp Bike.h Tricycle.h
   g++ -c main.cpp
# Custom command:
clean:
   rm -f GoRide *.o *~ # deleting executable, .o's and backups
```

Macro's

```
CC = g++
CFLAGS = -Wall
TARGET = GoRide
OBJECTS = Bike.o Tricycle.o main.o

# Linking all the object code:
all: $(OBJECTS)
   $(CC) $(FLAGS) $(OBJECTS) -o $(TARGET)
```

Special macro's

`CC`	Contains the current compiler. Defaults to cc
`CFLAGS`	Special options which are added to the built-in compile rule
`$@`	Full name of the current target.
`$?`	A list of files for current dependency which are out-of-date.
`$<`	The source file of the current (single) dependency.

Rules

```
%.o:  %.cpp
      g++ $< -Wall -o $@
```

Common errors

- Failing to put a TAB at the beginning of commands. This causes the commands not to run.
- To put a TAB at the beginning of a blank line. This causes the make utility to complain that there is a \blank" command.

Advanced common errors

- Using \ for continuation but not having the \ as the very last character of the line.
- Not including all dependencies.
- Listing object files in the wrong order.

Challenges

- Write a custom rule to tar the .cpp and .h files.
- Write a custom command that will print all .cpp files that have changed since the last build.
- Makefiles can also include files. Write a makefile that use a rule to generate the dependency list of the .cpp files in the current folder and include it automatically in the makfile.