WEEK-4 TUTORIAL

COURSE CODE: CSE231

COURSE NAME: OPERATING SYSTEM

MAKEFILE

- Setup and Installation
- Creating Simple Makefile
- Advance Features in Makefile

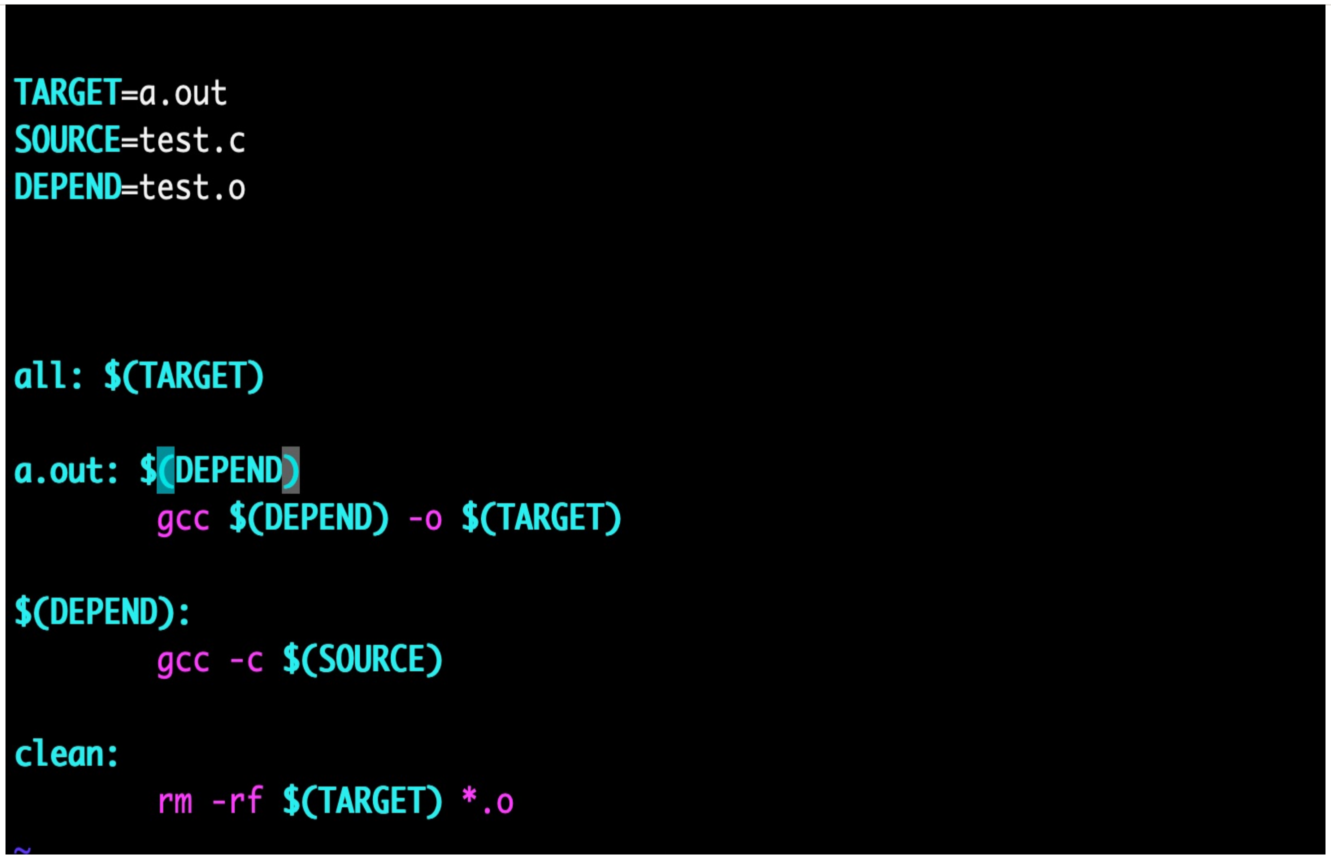
GNU MAKEFILE

- It makes compilation easy and predictable and also saves time and resources by compiling only the necessary files
- For Ubuntu, you need to run the following command for installing gnu make
  - sudo apt update **&&** sudo apt install build-essentials
- For Artix Linux
  - sudo pacman -S make

CREATING A SIMPLE MAKEFILE

- There are three basic things you have to know
  - Source files: The files are written in **.c / .cpp** you want to compile
  - Target files: The binary files are created after compilations
  - Dependencies: All the dependencies should exist before compilation

HOW MAKEFILE LOOKS LIKE

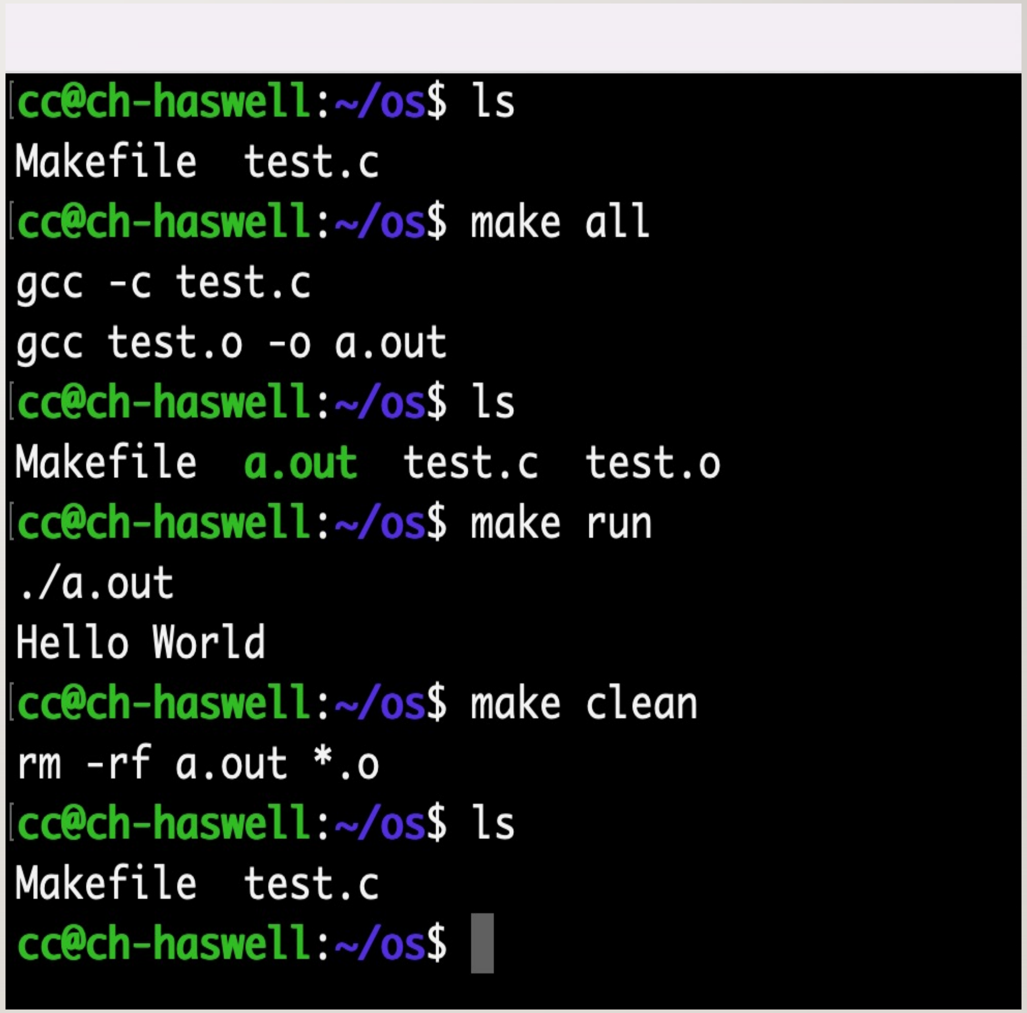


```
TARGET=a.out
SOURCE=test.c
DEPEND=test.o



all: $(TARGET)

a.out: $(DEPEND)
	gcc $(DEPEND) -o $(TARGET)

$(DEPEND):
	gcc -c $(SOURCE)

clean:
	rm -rf $(TARGET) *.o
~
```

Source: Find code in week-4-tutorial/makefile/code/v1

HOW TO RUN IT

- Compile your test.c file using Makefile
  - **make all** or **make**
  - you can see a.out binary file has been created
- Run a.out using Makefile
  - **make run**
  - It prints **Hello World** (output of test.c file)
- Clean / remove all binaries using Makefile
  - **make clean**
  - It removes all .o files and a.out



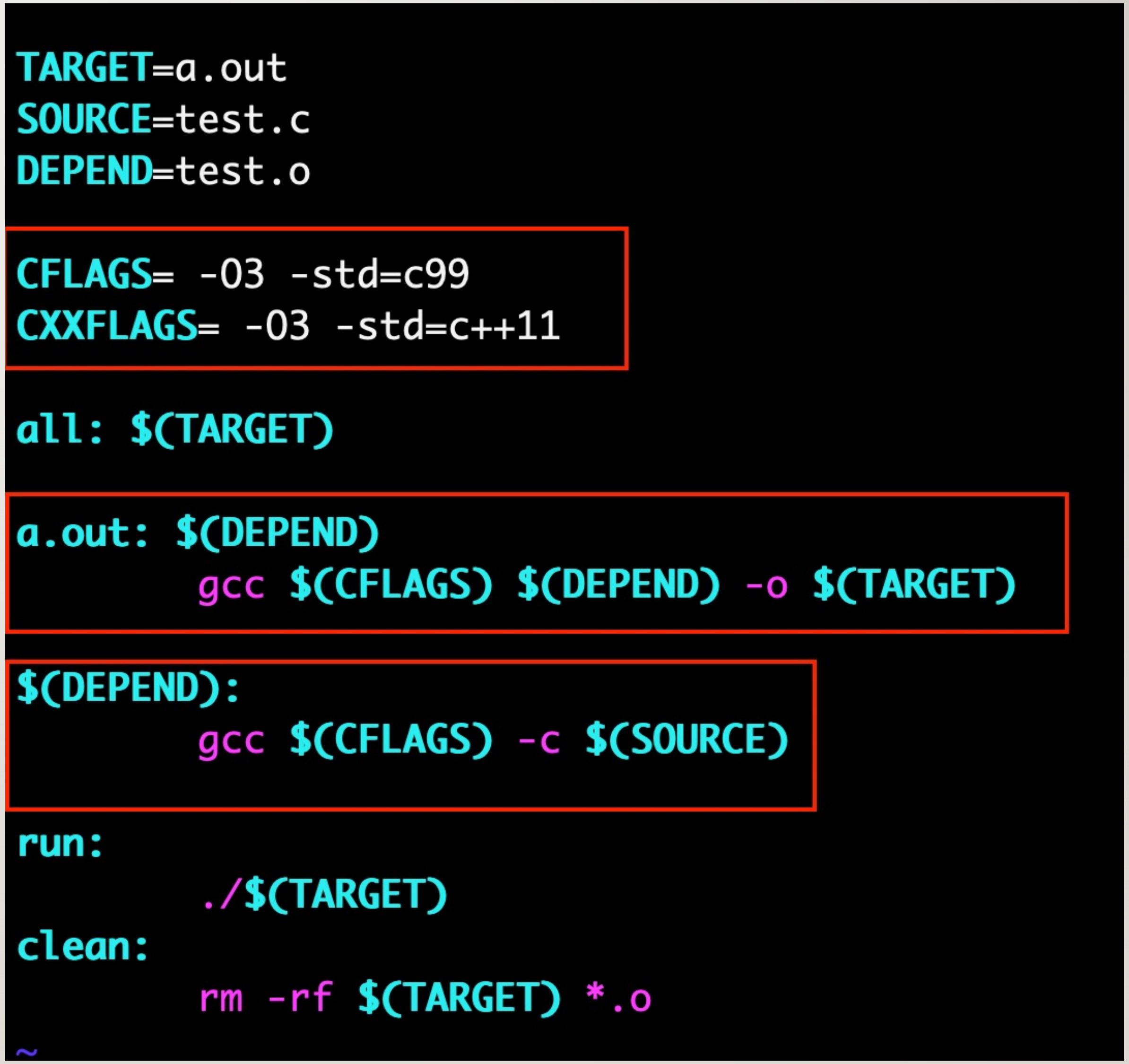
```
cc@ch-haswell:~/os$ ls
Makefile  test.c
cc@ch-haswell:~/os$ make all
gcc -c test.c
gcc test.o -o a.out
cc@ch-haswell:~/os$ ls
Makefile  a.out  test.c  test.o
cc@ch-haswell:~/os$ make run
./a.out
Hello World
cc@ch-haswell:~/os$ make clean
rm -rf a.out *.o
cc@ch-haswell:~/os$ ls
Makefile  test.c
cc@ch-haswell:~/os$
```

Source: Find code in week-4-tutorial/makefile/code/v1

ADVANCE TOPIC IN MAKEFILE

- Adding CFLAGS or CXXFLAGS and including header files (.h) using **-I** flag
- Creating assignment variables
- Creating a shared library
- Linking your code with a shared library using **-** L flag
- Automatic Variables in Makefile

ADDING FLAGS IN MAKEFILE



```
TARGET=a.out
SOURCE=test.c
DEPEND=test.o

CFLAGS= -O3 -std=c99
CXXFLAGS= -O3 -std=c++11

all: $(TARGET)

a.out: $(DEPEND)
	gcc $(CFLAGS) $(DEPEND) -o $(TARGET)

$(DEPEND):
	gcc $(CFLAGS) -c $(SOURCE)

run:
	./$(TARGET)
clean:
	rm -rf $(TARGET) *.o
~
```

Source: Find code in week-4-tutorial/makefile/code/v2

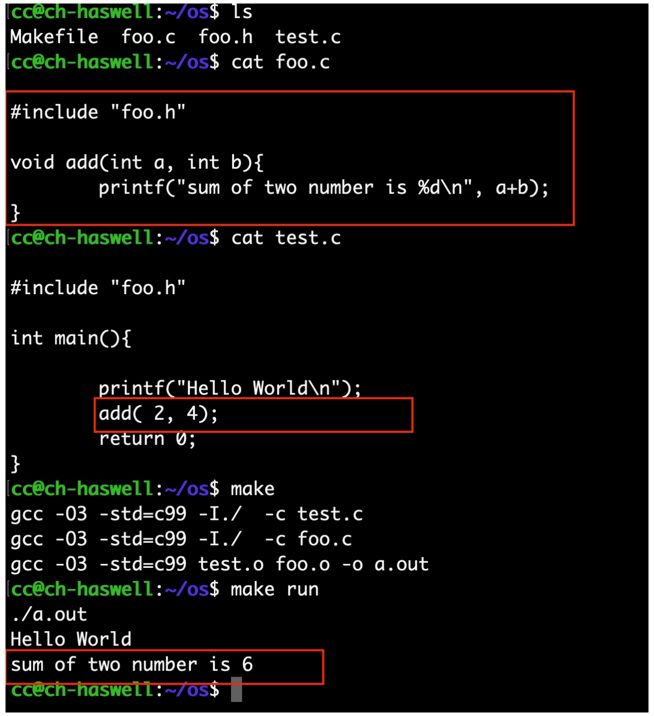
CREATING ASSIGNMENT VARIABLES

- There are four different assignment operators
  - Simple assignment (**:=**) : A simple assignment expression is evaluated only once, at the very first occurrence.
  - Recursive assignment (**=**) : A Recursive assignment expression is evaluated every time the variable is encountered in the code.
  - Conditional assignment (**?=**) : Conditional assignment assigns a value to a variable only if it does not have a value
  - Appending (**+=**) : It will append a new value to the previous value with space separated

INCLUDING HEADER FILES

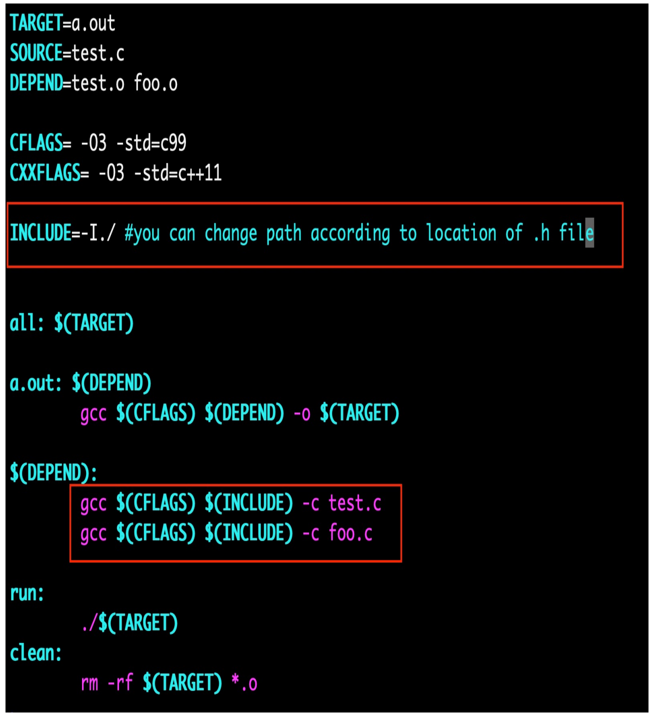
USING – I FLAG WE CAN INCLUDE HEADER FILES RESIDES IN DIFFERENT FOLDER BY PROVIDING PATH TO FOLDER

Console Output



```
cc@ch-haswell:~/os$ ls
Makefile  foo.c  foo.h  test.c
cc@ch-haswell:~/os$ cat foo.c

#include "foo.h"

void add(int a, int b){
        printf("sum of two number is %d\n", a+b);
}
cc@ch-haswell:~/os$ cat test.c

#include "foo.h"

int main(){

        printf("Hello World\n");
        add( 2, 4);
        return 0;
}
cc@ch-haswell:~/os$ make
gcc -O3 -std=c99 -I./  -c test.c
gcc -O3 -std=c99 -I./  -c foo.c
gcc -O3 -std=c99 test.o foo.o -o a.out
cc@ch-haswell:~/os$ make run
./a.out
Hello World
sum of two number is 6
cc@ch-haswell:~/os$
```

Changes in Makefile

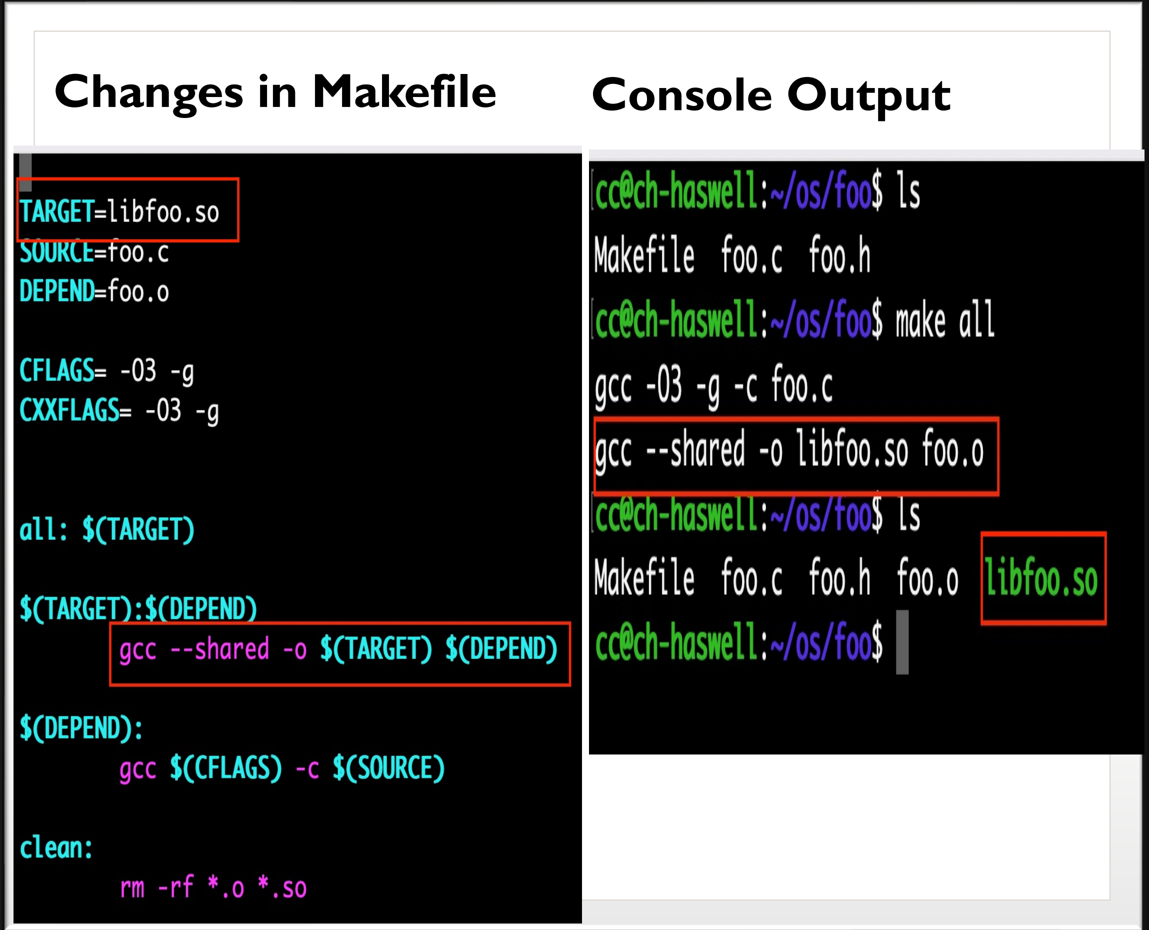


```
TARGET=a.out
SOURCE=test.c
DEPEND=test.o foo.o

CFLAGS= -O3 -std=c99
CXXFLAGS= -O3 -std=c++11

INCLUDE=-I./ #you can change path according to location of .h file


all: $(TARGET)

a.out: $(DEPEND)
	gcc $(CFLAGS) $(DEPEND) -o $(TARGET)

$(DEPEND):
	gcc $(CFLAGS) $(INCLUDE) -c test.c
	gcc $(CFLAGS) $(INCLUDE) -c foo.c

run:
	./$(TARGET)
clean:
	rm -rf $(TARGET) *.o
```

Source: Find code in week-4-tutorial/makefile/code/v3

CREATING SHARED LIBRARY (.SO)

- We also call it standalone libraries and you can call functions defined in libraries into your code using the linking mechanism
- Linking mechanism is covered in the next slide



Changes in Makefile

```
TARGET=libfoo.so
SOURCE=foo.c
DEPEND=foo.o

CFLAGS= -O3 -g
CXXFLAGS= -O3 -g


all: $(TARGET)

$(TARGET):$(DEPEND)
        gcc --shared -o $(TARGET) $(DEPEND)

$(DEPEND):
        gcc $(CFLAGS) -c $(SOURCE)

clean:
        rm -rf *.o *.so
```

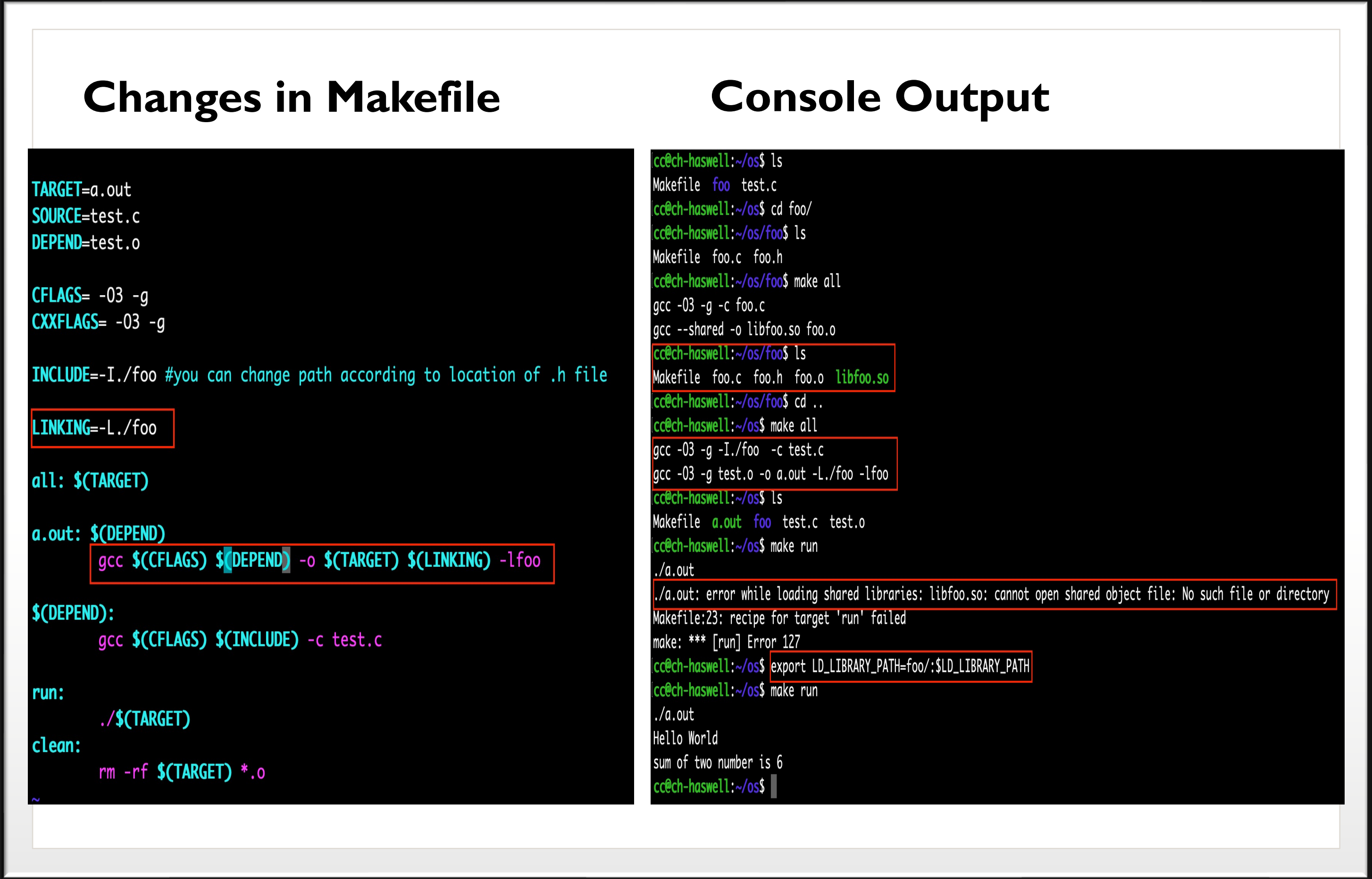
Console Output

```
cc@ch-haswell:~/os/foo$ ls
Makefile  foo.c  foo.h
cc@ch-haswell:~/os/foo$ make all
gcc -O3 -g -c foo.c
gcc --shared -o libfoo.so foo.o
cc@ch-haswell:~/os/foo$ ls
Makefile  foo.c  foo.h  foo.o  libfoo.so
cc@ch-haswell:~/os/foo$
```

Source: Find code in week-4-tutorial/makefile/code/v4/foo

LINKING SHARED LIBRARIES USING - L FLAG

SET **.SO** PATH TO **LD_LIBRARY_PATH** ENV VARIABLE AND MAKE LIBRARY AVAILABLE AT RUNTIME



Changes in Makefile

```
TARGET=a.out
SOURCE=test.c
DEPEND=test.o

CFLAGS= -O3 -g
CXXFLAGS= -O3 -g

INCLUDE=-I./foo #you can change path according to location of .h file

LINKING=-L./foo

all: $(TARGET)

a.out: $(DEPEND)
        gcc $(CFLAGS) $(DEPEND) -o $(TARGET) $(LINKING) -lfoo

$(DEPEND):
        gcc $(CFLAGS) $(INCLUDE) -c test.c

run:
        ./$(TARGET)
clean:
        rm -rf $(TARGET) *.o
```

Console Output

```
cc@ch-haswell:~/os$ ls
Makefile  foo  test.c
cc@ch-haswell:~/os$ cd foo/
cc@ch-haswell:~/os/foo$ ls
Makefile  foo.c  foo.h
cc@ch-haswell:~/os/foo$ make all
gcc -O3 -g -c foo.c
gcc --shared -o libfoo.so foo.o
cc@ch-haswell:~/os/foo$ ls
Makefile  foo.c  foo.h  foo.o  libfoo.so
cc@ch-haswell:~/os/foo$ cd ..
cc@ch-haswell:~/os$ make all
gcc -O3 -g -I./foo  -c test.c
gcc -O3 -g test.o -o a.out -L./foo -lfoo
cc@ch-haswell:~/os$ ls
Makefile  a.out  foo  test.c  test.o
cc@ch-haswell:~/os$ make run
./a.out
./a.out: error while loading shared libraries: libfoo.so: cannot open shared object file: No such file or directory
Makefile:23: recipe for target 'run' failed
make: *** [run] Error 127
cc@ch-haswell:~/os$ export LD_LIBRARY_PATH=foo/:$LD_LIBRARY_PATH
cc@ch-haswell:~/os$ make run
./a.out
Hello World
sum of two number is 6
cc@ch-haswell:~/os$
```

Source: Find code in week-4-tutorial/makefile/code/v4

AUTOMATIC VARIABLES IN MAKEFILE

- $@ : The filename representing the target.
- $< : The filename of the first prerequisite (all dependencies).
- $? : The names of all prerequisites that are newer than the target, separated by spaces.
- $^ : The filenames of all the prerequisites, separated by spaces. This list has duplicate filenames removed since for most uses, such as compiling, copying, etc., duplicates are not wanted.

Source: Find code in week-4-tutorial/makefile/code/v5

REFERENCE MATERIAL

- https://www.gnu.org/software/make/manual/html_node/Concept-Index.html#Concept-Index
- https://stackoverflow.com/questions/4879592/whats-the-difference-between-and-in-makefile
- https://stackoverflow.com/questions/3220277/what-do-the-makefile-symbols-and-mean
- https://earthly.dev/blog/g++-makefile/
- https://iq.opengenus.org/create-shared-library-in-cpp/

DEMO