

**Institute of Computer Technology**  
**B. Tech. Computer Science and Engineering**

**Semester: IV**

**Sub: Operating System**

**Course Code: 2CSE402**

## Practical Number:1

### Objective:

*Basic Linux Command Practice.*

Q.1: Makefiles are used to help decide which parts of a large program need to be recompiled. In the vast majority of cases, C or C++ files are compiled. Other languages typically have their own tools that serve a similar purpose as Make. It can be used beyond programs too, when you need a series of instructions to run depending on what files have changed. This experiment focus on the C/C++ compilation use case attached herewith. Here's an example dependency graph that you might build with Make. If any file's dependencies changes, then the file will get recompiled.

in short, A makefile is a special file that lists a set of rules for compiling a project. These rules include , which can be an action make needs to take (eg. "clean" or "build") or the files/objects make will need to build (eg. .o files or an executable), and the commands that need to be run in order to build that target. When you call the program make, it runs through each of these targets in your Makefile and executes them.

Sample Code:

```
blah.o: blah.c
```

```
gcc -c blah.c -o blah.o # Runs second
```

```
blah.c:
```

```
echo "int main() { return 0; }" > blah.c # Runs first
```

Tasks to be done by students:

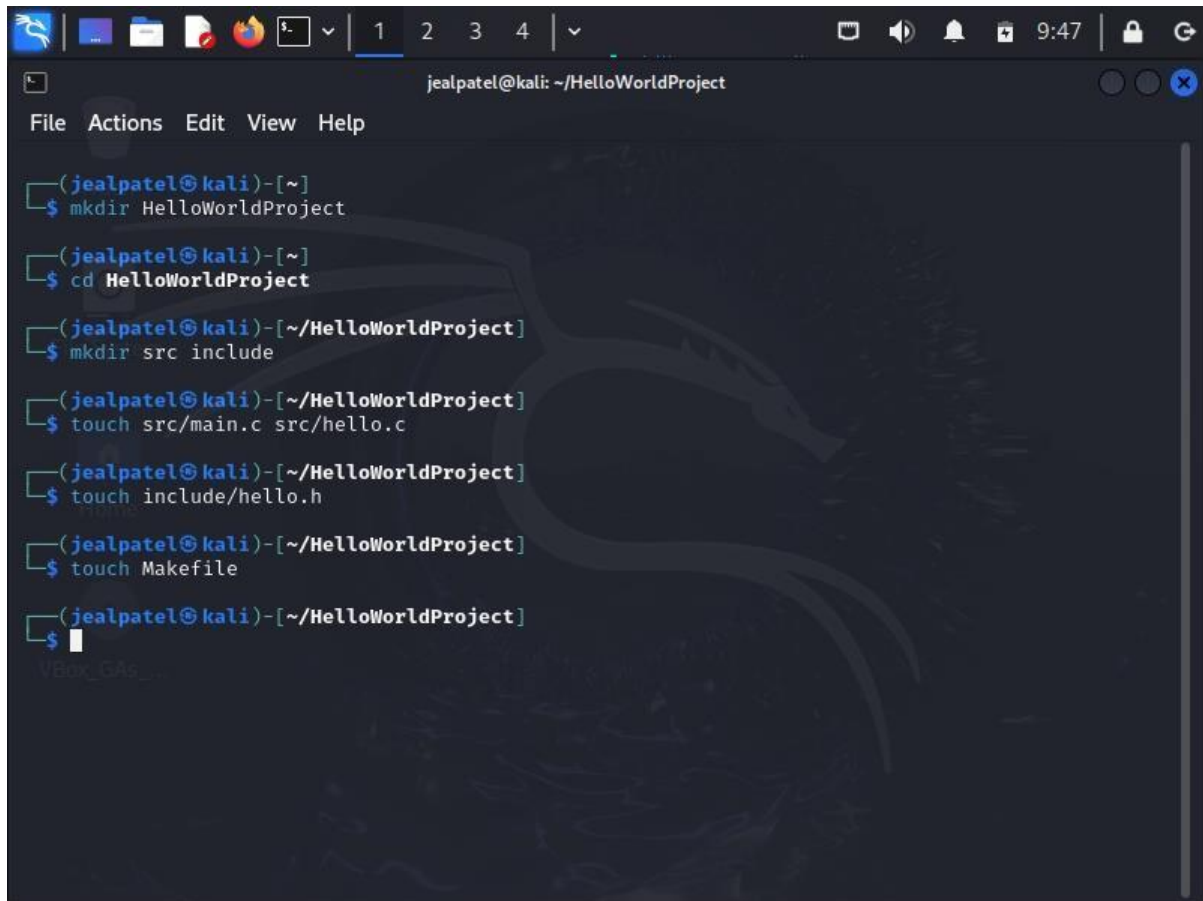
1. Write a simple “Hello World” program. Move the part that outputs the

text (i.e., the `printf()` function call), into a separate file `hello.c`. Edit an appropriate interface file `hello.h` in order to use the subroutine in the `main.c` program. Write a Makefile that maintains the program structure. Use a variable `CC` in the makefile to define the compiler, e.g., `CC=gcc`.

2. Place the header files and C/C++ files in separate directory and compile and execute those files using Makefile Utility.

### Steps :

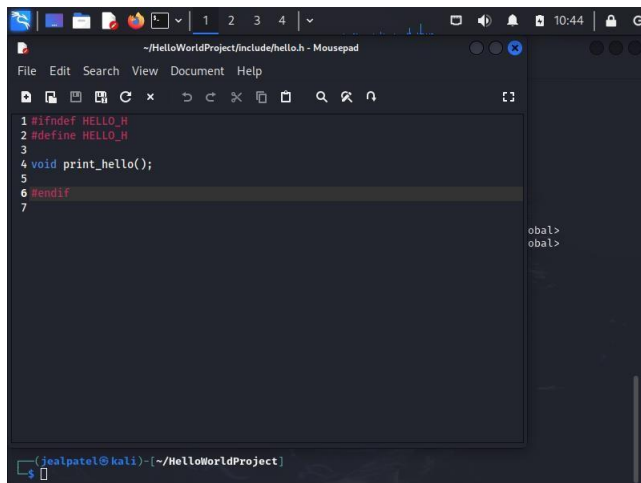
*Step 1: Create Files and Directories*

A screenshot of a terminal window titled 'jealpatel@kali: ~/HelloWorldProject'. The terminal shows a series of commands being executed to create a project directory structure. The commands are: 'mkdir HelloWorldProject', 'cd HelloWorldProject', 'mkdir src include', 'touch src/main.c src/hello.c', 'touch include/hello.h', and 'touch Makefile'. The prompt is '(jealpatel@kali)-[~/HelloWorldProject]' for each command. The terminal has a dark background with a Kali Linux dragon logo watermark. The window's title bar shows standard Linux window controls and system icons on the right, including a clock showing 9:47.

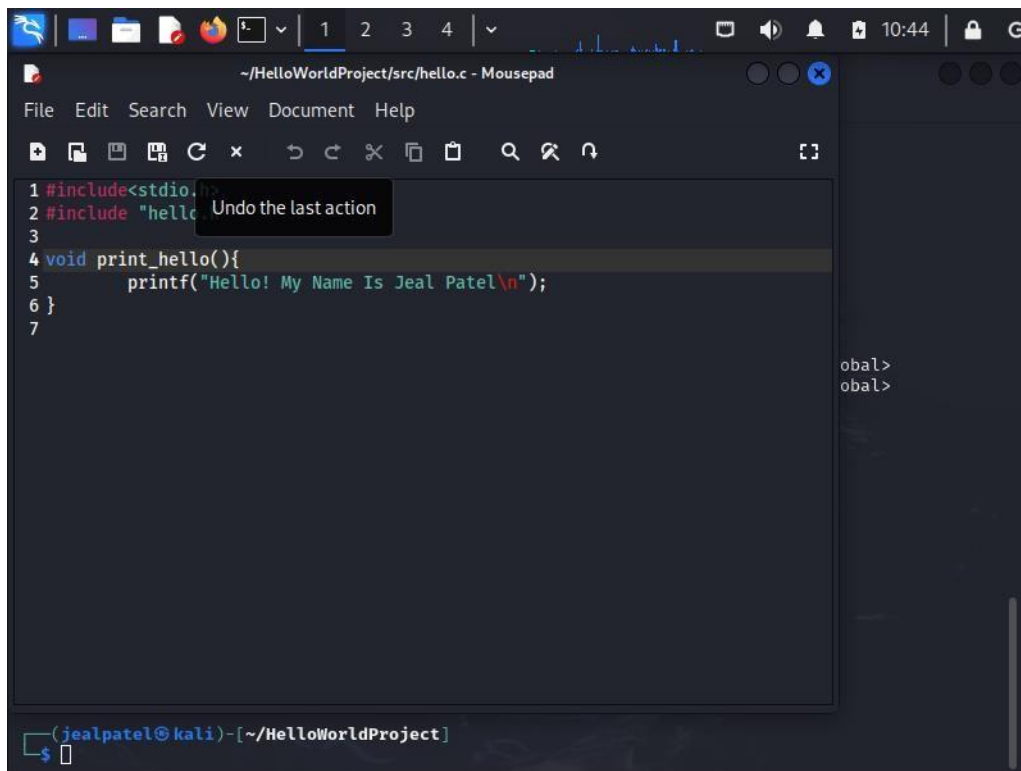
```
(jealpatel@kali)-[~]  
$ mkdir HelloWorldProject  
  
(jealpatel@kali)-[~]  
$ cd HelloWorldProject  
  
(jealpatel@kali)-[~/HelloWorldProject]  
$ mkdir src include  
  
(jealpatel@kali)-[~/HelloWorldProject]  
$ touch src/main.c src/hello.c  
  
(jealpatel@kali)-[~/HelloWorldProject]  
$ touch include/hello.h  
  
(jealpatel@kali)-[~/HelloWorldProject]  
$ touch Makefile  
  
(jealpatel@kali)-[~/HelloWorldProject]  
$
```

*Step 2: Write Code Into Specific Files*

1. `nano include/hello.h`



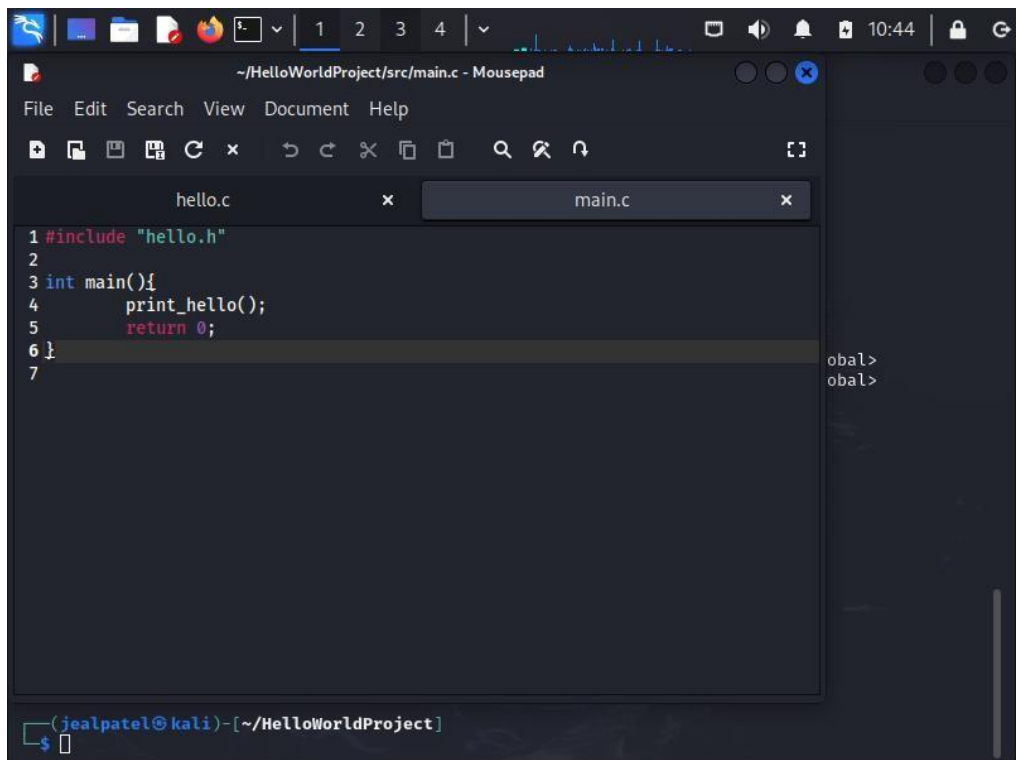
## 2. nano src/hello.c



```
~/HelloWorldProject/src/hello.c - Mousepad
File Edit Search View Document Help
1 #include<stdio.h>
2 #include "hello.h"
3
4 void print_hello(){
5     printf("Hello! My Name Is Jeal Patel\n");
6 }
7

(jealpatel@kali) - [~/HelloWorldProject]
$
```

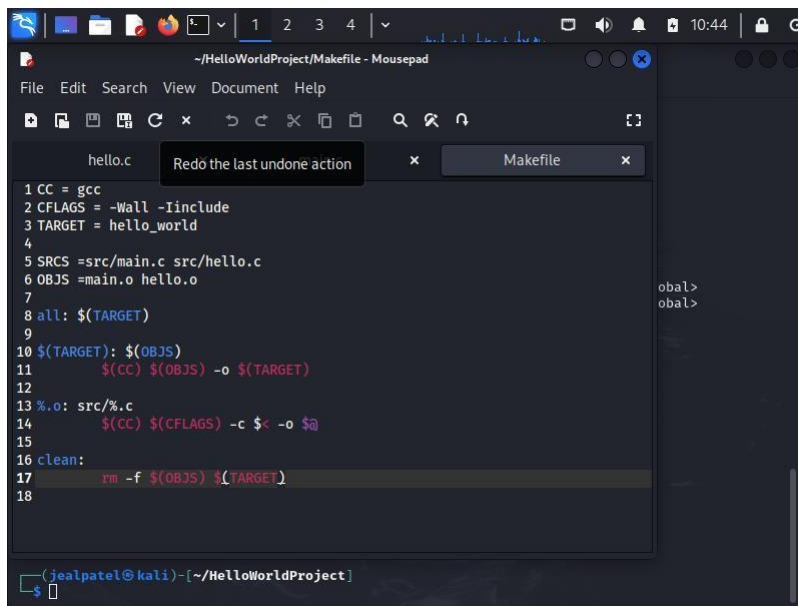
## 3. nano src/main.c



```
~/HelloWorldProject/src/main.c - Mousepad
File Edit Search View Document Help
hello.c x main.c x
1 #include "hello.h"
2
3 int main(){
4     print_hello();
5     return 0;
6 }
7

(jealpatel@kali) - [~/HelloWorldProject]
$
```

#### 4. nano Makefile

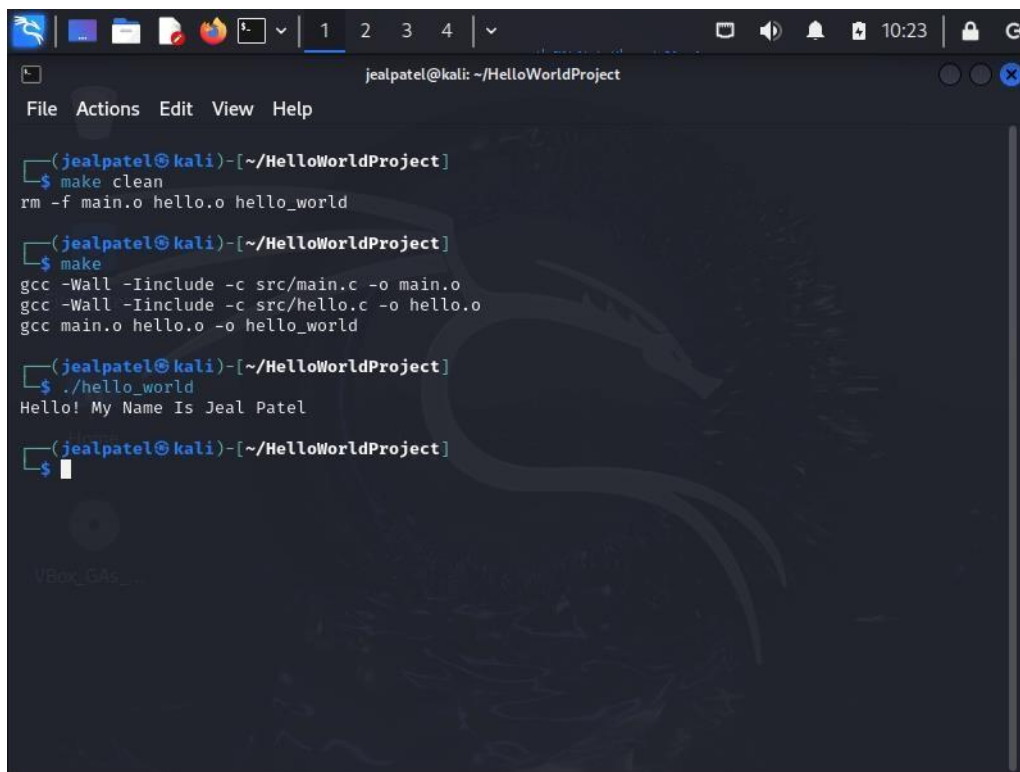


The screenshot shows a nano text editor window titled '~/.HelloWorldProject/Makefile - Mousepad'. The editor contains a Makefile with the following content:

```
1 CC = gcc
2 CFLAGS = -Wall -Iinclude
3 TARGET = hello_world
4
5 SRCS =src/main.c src/hello.c
6 OBJS =main.o hello.o
7
8 all: $(TARGET)
9
10 $(TARGET): $(OBJS)
11     $(CC) $(OBJS) -o $(TARGET)
12
13 %.o: src/%.c
14     $(CC) $(CFLAGS) -c $< -o $@
15
16 clean:
17     rm -f $(OBJS) $(TARGET)
18
```

Below the editor, a terminal window shows the command prompt: `(jealpatel@kali)-[~/HelloWorldProject]` with a cursor on the line.

#### Step 3: Compile the program



The screenshot shows a terminal window titled 'jealpatel@kali: ~/.HelloWorldProject'. The terminal displays the following commands and output:

```
(jealpatel@kali)-[~/HelloWorldProject]
$ make clean
rm -f main.o hello.o hello_world

(jealpatel@kali)-[~/HelloWorldProject]
$ make
gcc -Wall -Iinclude -c src/main.c -o main.o
gcc -Wall -Iinclude -c src/hello.c -o hello.o
gcc main.o hello.o -o hello_world

(jealpatel@kali)-[~/HelloWorldProject]
$ ./hello_world
Hello! My Name Is Jeal Patel

(jealpatel@kali)-[~/HelloWorldProject]
$
```