Operating Systems Lab

Experiment 01: Basic Linux Commands

1. ls – It lists the contents of the directory.

```
localhost:~# ls
bench.py    hello.c     hello.js    readme.txt
```

2. cat – It displays the contents of the files

```
localhost:~# cat hello.c
#include <stdio.h>
int main(void)
{
        printf("hello world\n");
        return 0;
}
```

3. pwd – It prints the current working directory

```
localhost:~# pwd
/root
```

4. whoami – It displays the current user name

```
localhost:~# whoami
root
```

5. clear – It clears the terminal screen.

```
localhost:~#
```

6. mkdir – It creates a new directory

```
localhost:~# mkdir Rohan
localhost:~# ls
 Rohan       bench.py    hello.c     hello.js    readme.txt
```

7. cd – It changes the current working directory

```
localhost:~# cd Rohan
localhost :/Rohan#
```

8. cd .. – It moves back to the parent directory.

```
localhost: /Rohan#    cd ..
localhost:~#
```

9. rmdir – It deletes the directory.

```
localhost:~# rmdir Rohan
localhost:~# ls
bench.py    hello.c     hello.js    readme.txt
```

10. cat > helllo.txt – It allows us to create a file named hello.txt and allows to write content in it. Press Ctrl + D to save and exit.

```
localhost:~# cat > hello.txt
This is my 1st OS lab practical.
localhost:~# ls
bench.py    hello.c     hello.js    hello.txt   readme.txt
localhost:~# cat hello.txt
This is my 1st OS lab practical.
```

11. cat >> hello.txt – It appends the content to the hello.txt file.

```
localhost:~# cat >> hello.txt
OS stands for Operating Systems...
localhost:~# ls
bench.py    hello.c     hello.js    hello.txt   readme.txt
localhost:~# cat hello.txt
This is my 1st OS lab practical.
OS stands for Operating Systems...
```

12. cp hello.txt bye.txt – It copies hello.txt file to a new file named bye.txt. Here, hello.txt acts as source file and bye.txt as a destination file.

```
localhost:~# cp hello.txt bye.txt
localhost:~# ls
bench.py    bye.txt     hello.c     hello.js    hello.txt   readme.txt
localhost:~# cat bye.txt
This is my 1st OS lab practical.
OS stands for Operating Systems...
```

13. mv bye.txt file.txt – It renames or moves bye.txt file to file.txt file. Here, bye.txt is source file and file.txt is destination file.

```
localhost:~# mv bye.txt file.txt
localhost:~# ls
bench.py    file.txt    hello.c     hello.js    hello.txt
localhost:~# cat file.txt
This is my 1st OS Lab practical
OS stands for Operating Systems...
```

14. rm file.txt – It deletes the file 'file.txt'.

```
localhost:~# rm file.txt
localhost:~# ls
bench.py    hello.c     hello.js    hello.txt   readme.txt
localhost:~#
```

Experiment 02

Create an Animal directory, now create 2 more directories in Animal namely Mammal and Reptile. In Mammal directory create 2 files cow.txt and lizard.txt with 2 lines about cow and lizard respectively. Now, move lizard.txt from mammal to reptile directory.

Step 1: Create a directory Animal

mkdir Animal

```
localhost:~# mkdir Animal
localhost:~# ls
Animal      bench.py    hello.c
```

Step 2: In Animal, create two directories mammals and reptiles

cd Animal
mkdir mammals reptiles

```
localhost:~# cd Animal
localhost:~/Animal# mkdir mammals reptiles
localhost:~/Animal# ls
mammals   reptiles
```

Step 3: In mammals add a file cow.txt with 2 lines on cow written in the file.

cd mammals
cat > cow.txt

```
localhost:~/Animal# cd mammals
localhost:~/Animal/mammals# cat > cow.txt
Cows are a source of milk.
Cows are herbivorous animals.
```

Step 4: Again, in mammals add a file lizard.txt with 2 lines on lizard written in the file.

cat > lizard.txt

```
localhost:~/Animal/mammals# cat > lizard.txt
Lizards are cold-blooded creatures.
Lizards have long tails.
```

Step 5: Now, move lizard.txt file to reptiles from mammals.

mv lizard.txt ../reptiles/lizard.txt

```
localhost:~/Animal/mammals# mv lizard.txt ../reptiles/lizard.txt
localhost:~/Animal/mammals# ls
cow.txt
localhost:~/Animal/mammals# cd ..
localhost:~/Animal# cd reptiles
localhost:~/Animal/reptiles# ls
lizard.txt
```

Experiment 03

Aim: Create a directory vehicle. In Vehicle create three directories with names Fourwheelers, Threewheelers, Twowheelers. In Fourwheelers create directories Bus, Car, Truck. In Threewheelers create directories Auto. In Twowheelers create directories cycle and scooty. In Car directory create files carbrand.txt, busbrand.txt, truckband.txt, autocolor.txt, cyclebrand.txt. Write two lines in each file. Move each files in their respective directories, also delete cycle directory finally.

Procedure:

Step 1: Create a directory vehicle (mkdir vehicle)

```
localhost:~# ls
bench.py    hello.c     hello.js    readme.txt
localhost:~# mkdir vehicle
localhost:~# ls
bench.py    hello.c     hello.js    readme.txt  vehicle
```

Step 2: Change to directory vehicle (cd vehicle)

```
localhost:~# cd vehicle
```

Step 3: Create to sub directories in vehicle (mkdir Fourwheelers Threewheelers Twowheelers)

```
localhost:~/vehicle# mkdir Fourwheelers Threewheelers Twowheelers
localhost:~/vehicle# ls
Fourwheelers   Threewheelers  Twowheelers
```

Step 4 : Change to directory Fourwheelers (cd Fourwheelers)

```
localhost:~/vehicle# cd Fourwheelers
```

Step 5 : Create to sub directories in vehicle (mkdir bus car truck)

```
localhost:~/vehicle/Fourwheelers# mkdir bus car truck
localhost:~/vehicle/Fourwheelers# ls
bus    car    truck
```

Step 6: Come to vehicle directory and open Threewheelers directory and create a Auto Directory.

```
localhost:~/vehicle/Fourwheelers# cd
localhost:~# cd vehicle
localhost:~/vehicle# cd Threewheelers
localhost:~/vehicle/Threewheelers# mkdir Auto
localhost:~/vehicle/Threewheelers# ls
Auto
```

Step 7: Come to vehicle directory and open Twowheelers directory and create a Cycle and scooty Directory.

```
localhost:~/vehicle# cd Twowheelers
localhost:~/vehicle/Twowheelers# mkdir cycle scooty
localhost:~/vehicle/Twowheelers# ls
cycle   scooty
```

Step 8: In Car directory create files carbrand.txt, busbrand.txt, truckband.txt, autocolor.txt, cyclebrand.txt. and write two lines in each file using cat command.

```
localhost:~/vehicle/Fourwheelers/car# cat > carbrand.txt
Car brands Consist of Toyota,Maruti Suzuki.
More can be Skoda,Kia.
localhost:~/vehicle/Fourwheelers/car# cat > busbrand.txt
Bus Brands Consist of
Force,Mercedes,Volvo
localhost:~/vehicle/Fourwheelers/car# cat > truckbrand.txt
Truck Brand Consist of
Tata,Bharat Benz
localhost:~/vehicle/Fourwheelers/car# cat > autocolor.txt
Auto Comes in many colours such as
Yellow,Pink,Black,Blue,Green
localhost:~/vehicle/Fourwheelers/car# cat > cyclebrand.txt
Cycle brands are
avon,hero and many more
localhost:~/vehicle/Fourwheelers/car# ls
autocolor.txt   busbrand.txt    carbrand.txt    cyclebrand.txt  truckbrand.txt
```

Step 9: Now move the files respective to their directory locations.

```
localhost:~/vehicle/Fourwheelers/car# mv busbrand.txt ..//bus/busbrand.txt
localhost:~/vehicle/Fourwheelers/car# ls
autocolor.txt   carbrand.txt    cyclebrand.txt  truckbrand.txt
```

```
localhost:~/vehicle/Fourwheelers/car# mv truckbrand.txt ..//truck/truckbrand.txt
```

```
localhost:~/vehicle/Fourwheelers/car# mv autocolor.txt ..//..//Auto
localhost:~/vehicle/Fourwheelers/car# ls
carbrand.txt    cyclebrand.txt
```

```
localhost:~/vehicle/Fourwheelers/car# mv cyclebrand.txt ..//..//cycle
localhost:~/vehicle/Fourwheelers/car# ls
carbrand.txt
```

Step 10: Now check the files with respective to their directories

```
localhost:~/vehicle/Fourwheelers# ls
bus     car     truck
localhost:~/vehicle/Fourwheelers# cd bus
localhost:~/vehicle/Fourwheelers/bus# ls
busbrand.txt
localhost:~/vehicle/Fourwheelers/bus# cd ..
localhost:~/vehicle/Fourwheelers# cd car
localhost:~/vehicle/Fourwheelers/car# ls
carbrand.txt
localhost:~/vehicle/Fourwheelers/car# cd ..
localhost:~/vehicle/Fourwheelers# cd truck
localhost:~/vehicle/Fourwheelers/truck# ls
truckbrand.txt
```

Step 11: Now delete the cycle directory.

```
localhost:~/vehicle/Twowheelers# rmdir cycle
localhost:~/vehicle/Twowheelers# ls
scooty
```

Experiment 06

Aim: Study the usage of for loop in shell

- Using a for loop in shell scripting can be handy for iterating through lists of items or performing operations on files. In shell scripting, for loops typically follow this syntax:

```
for item in list
do
    # commands to be executed for each item
done
```

1. Echo Basic Message

```
#!/bin/bash

SERVERS="s1 s2 s3"
for S in $SERVERS; do
    echo "Updating pkg on: $S"
done
```

```
#!/bin/bash

SERVERS="s1 s2 s3"
for S in $SERVERS; do
    echo "Updating pkg on: $S"
done

localhost:~/aatif# bash test1.sh
Updating pkg on: s1
Updating pkg on: s2
Updating pkg on: s3
```

2. Iterate Range of Numbers

```
#!/bin/bash
for value in {1..20}
do
   echo "Number: $value"
done
```

```
#!/bin/bash

for value in {1..20}
do
    echo "Number: $value"
done
```

Output:

```
localhost:~/aatif# bash test2.sh
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Number: 6
Number: 7
Number: 8
Number: 9
Number: 10
Number: 11
Number: 12
Number: 13
Number: 14
Number: 15
Number: 16
Number: 17
Number: 18
Number: 19
Number: 20
```

3. Iterate on Multiple Files

```
#!/bin/bash

for file in /root/*
do
   chmod 755 "$file"
   echo "Updated permission for: $file"
done
```

```
#!/bin/bash

for file in /root/*
do
    chmod 755 "$file"
    echo "Updated permission for: $file"
done
localhost:~/aatif# bash test3.sh
Updated permission for: /root/aatif
Updated permission for: /root/bench.py
Updated permission for: /root/hello.c
Updated permission for: /root/hello.js
Updated permission for: /root/readme.txt
```

4. Create Infinite Loop

```
#!/bin/bash

for (( ; ; ))
do
   echo "This is an infinite loop"
   echo "Use Ctrl+C to stop it"
done
```

```
#!/bin/bash

for (( ; ; ))
do
    echo "This is an infinite loop"
    echo "Use Ctrl+C to stop it"
done

localhost:~/aatif# bash test4.sh
This is an infinite loop
Use Ctrl+C to stop it
This is an infinite loop
Use Ctrl+C to stop it
```

5. Nested for Loop

```
#!/bin/bash

for server in Apache DB
do
   for app in apache php do
   do
      echo "$server can run $app LAMP package"
   done
done
```

```
#!/bin/bash

for server in Apache DB
do
    for app in apache php do
    do
        echo "$server can run $app LAMP package"
    done
done
localhost:~/aatif# bash test5.sh
Apache can run apache LAMP package
Apache can run php LAMP package
Apache can run do LAMP package
DB can run apache LAMP package
DB can run php LAMP package
DB can run do LAMP package
```

6. Use Array in for Loop

```
#!/bin/bash

apps=("apache" "mysql" "php")
for app in "${apps[@]}"
do
    echo "The application name is: $app"
done
```

```
localhost:~/aatif# bash test6.sh
The application name is: apache
The application name is: mysql
The application name is: php
```

7. Use Break in for Loop

```
#!/bin/bash

for file in ~/.* ; do
    if [[ "$file" == "./bash.sh" ]]
    then
        echo "$file is available"
        break
    fi
done
```

```
localhost:~/aatif# bash test7.sh
```

8. Use Command Substitution

```
#!/bin/bash

for log in $(cat ~/testfile)
do
    echo "Log entry: $log"
done
```

```
localhost:~/aatif# bash test8.sh
```

Experiment 10

Aim: Program to create orphan process and zombie process.

An orphan process is a process whose parent has terminated before it finishes its execution.

A zombie process is a process that has completed execution but still has an entry in the process table.

1) Orphan Process

vi orphan.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{ pid_t p;
   p = fork();
   if (p == 0)
   { sleep(5);
      printf("I am child having PID: %d\n",
getpid());
      printf("My parent pid is: %d\n",
getppid()); }
   else
   { printf("I am parent having pid: %d\n",
getpid());
      printf("My child pid is: %d\n", p);
   }
}
```

:wq

```
gcc -o orphan orphan.c

./orphan
```

```
I am parent having pid : 81
My child pid is : 82
localhost:~/aatif# I am child having PID : 82
My parent pid is : 1
```

2) Zombie Process

vi zombie.c

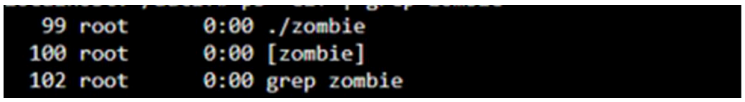
```
#include <stdio.h>
#include <unistd.h>
int main()
{pid_t p;
   p = fork();
   if (p == 0)
   {printf("Child having id : %d\n",
getpid());}
   else
   {printf("Parent having id : %d\n",
getpid());
       sleep(15); // run the ps command
during this time.}
}
```

:wq

```
gcc -o zombie zombie.c

./zombie &

ps -elf | grep defunct
```



Experiment 11

Aim: Write a program to create threads in Linux

vi thread.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
void *thread_function(void
*arg);
int i, j;
int main()
{pthread_t a_thread;
   pthread_create(&a_thread,
NULL, thread_function,
NULL);
   pthread_join(a_thread,
NULL);
   printf("Inside main
program\n");
   for(j = 20; j <= 25; j++)
   {printf("j : %d\n", j);
      sleep(1);}}
void *thread_function(void
*arg)
{printf("Inside thread\n");
   for(i = 0; i < 5; i++)
   {printf("i : %d\n", i);
      sleep(1);}}
```

```
inside thread
i : 0
i : 1
i : 2
i : 3
i : 4
Inside main program
j : 20
j : 21
j : 22
j : 23
j : 24
```

:wq

```
gcc thread.c -o thread -lpthread

./thread
```

Experiment 12

Aim: Write a program for Process Synchronization using mutex lock.

vi mutex.c

```
#include<pthread.h>
#include<stdio.h>
#include<unistd.h>
void *fun1();
void *fun2();
int shared=1;
pthread_mutex_t l;
int main(){
pthread_t thread1,thread2;
pthread_mutex_init(&l,NULL);
pthread_create(&thread1,NULL,fun1,NULL);
pthread_create(&thread2,NULL,fun2,NULL);
pthread_join(thread1,NULL);
pthread_join(thread2,NULL);
printf("Final value of shared is: %d\n",shared);
pthread_mutex_destroy(&l);
return 0;}
void *fun1(){
int x;
printf("Thread1 trying to acquire lock\n");
pthread_mutex_lock(&l);
printf("Thread1 acquired lock\n");
x=shared;
printf("Thread1 reads the shared variable as:
%d\n",x);
x++;
printf("Local updation by Thread1: %d\n",x);
sleep(1);
shared=x;
printf("Value of shared variable updated by
Thread1 is: %d\n",shared);
pthread_mutex_unlock(&l);
printf("Thread1 released the lock\n");
return NULL;}
void *fun2(){
int x;
printf("Thread2 trying to acquire lock\n");
pthread_mutex_lock(&l);
printf("Thread2 acquired lock\n");
x=shared;
printf("Thread2 reads the shared variable as:
%d\n",x);x++;
printf("Local updation by Thread2: %d\n",x);
sleep(1);
shared=x;
printf("Value of shared variable updated by
Thread2 is: %d\n",shared);
pthread_mutex_unlock(&l);
printf("Thread2 released the lock\n");
return NULL;}
:wq
```

```
#include<pthread.h>
#include<stdio.h>
#include<unistd.h>
void *fun1();
void *fun2();
int shared=1;
pthread_mutex_t l;
int main(){
pthread_t thread1,thread2;
pthread_mutex_init(&l,NULL);
pthread_create(&thread1,NULL,fun1,NULL);
pthread_create(&thread2,NULL,fun2,NULL);
pthread_join(thread1,NULL);
pthread_join(thread2,NULL);
printf("Final value of shared is: %d\n",shared);
pthread_mutex_destroy(&l);
return 0;
}
void *fun1(){
int x;
printf("Thread1 trying to acquire lock\n");
pthread_mutex_lock(&l);
printf("Thread1 acquired lock\n");
x=shared;
printf("Thread1 reads the shared variable as: %d\n",x);
x++;
printf("Local updation by Thread1: %d\n",x);
sleep(1);
shared=x;
printf("Value of shared variable updated by Thread1 is: %d\n",shared);
pthread_mutex_unlock(&l);
printf("Thread1 released the lock\n");
return NULL;
}
void *fun2(){
int x;
printf("Thread2 trying to acquire lock\n");
pthread_mutex_lock(&l);
printf("Thread2 acquired lock\n");
x=shared;
printf("Thread2 reads the shared variable as: %d\n",x);
x++;
printf("Local updation by Thread2: %d\n",x);
sleep(1);
shared=x;
printf("Value of shared variable updated by Thread2 is: %d\n",shared);
pthread_mutex_unlock(&l);
printf("Thread2 released the lock\n");
return NULL;
}
```

```
Thread2 trying to acquire lock
Thread2 acquired lock
Thread2 reads the shared variable as: 1
Local updation by Thread2: 2
Thread1 trying to acquire lock
Value of shared variable updated by Thread2 is: 2
Thread2 released the lock
Thread1 acquired lock
Thread1 reads the shared variable as: 2
Local updation by Thread1: 3
Value of shared variable updated by Thread1 is: 3
Thread1 released the lock
Final value of shared is: 3
```

```
gcc mutex.c -o mutex -lpthread
./mutex
```

Experiment 04

Aim: Learn use of chmod command and vi text editor.

The “chmod” command modifies the read, write, and execute permissions of specified files. The octal digits used for assigning permissions are as follows:

Octal Digit	Permissions	Symbolic Display
7	read, write, execute	rwx
6	read, write	rw-
5	read, execute	r-x
4	read	r--
3	write, execute	-wx
2	write	-w-
1	execute	--x
0	no permissions	---

- Owner is denoted by ‘u’.
- Group is denoted by ‘g’.
- Others is denoted by ‘o’.

Examples
1. Using octal notation:
a)chmod 711 test.sh
2. Using symbolic notation:
a)chmod u+rwx test.sh
b)chmod go+--x test.sh
The “vi text editor” is a powerful text editor available in Unix and Linux systems. It is widely used for editing configuration files and scripts.

Basic Modes in vi

1. Insert Mode (For writing text)
a. Press i to enter insert mode.
b. Start typing the content.

2. Command Mode (Default mode)
a. Used for navigation and executing commands.
b. Press Esc to return to this mode.

3. Last Line Mode (For saving and exiting)
a. Press Esc, then type : to enter last line mode.
b. :wq → Save and exit.

Program Case 1: Creating hello.sh file in a directory with your name and giving permission to user only using chmod command.

ls
mkdir Aatif
cd Aatif
cat > hello.sh
#!/bin/bash
echo “Hello World”
chmod u+rwx hello.sh
./hello.sh

```
localhost:~# ls
bench.py    hello.c     hello.js    readme.txt
localhost:~# mkdir Aatif
localhost:~# cd Aatif
localhost:~/Aatif# cat > hello.sh
#!/bin/bash
echo"Hello World"
```

Program Case 2: Creating vi text editor file namely test.sh
vi test.sh
i (for start inserting text)
#!/bin/bash
var1=”hello”
var2=”Aatif”
echo $var1 $var2
esc (for escaping insert mode and come back to command mode)
:wq(for save and exit)
chmod u+rwx test.sh
./test.sh

```
localhost:~# vi test.sh
#!/bin/bash
var1="Hello"
var2="Aatif"
echo $var1 $var2
~
~
localhost:~# chmod u+rwx test.sh
localhost:~# ./test.sh
```

Experiment 05

Aim: if-else statement in bash

Bash scripting allows conditional execution using if-else statements. These statements enable decision-making within a script, executing different commands based on whether a condition evaluates to true or false.

1. Simple if statement

```
#!/bin/bash
if [ 1 -eq 1 ];
then
   echo "1 is equal to 1"
fi
```

Save as: testif.sh

```
localhost:~/Aatif# cat>testif1.sh
#!/bin/bash
if [ 1 -eq 1 ];
then
    echo "1 is equal to 1"
fi
```

2. if-else statement

```
#!/bin/bash
if [ 1 -eq 1 ];
then
   echo "1 is equal to 1"
else
   echo "The numbers are not equal"
fi
```

Saveas:testif2.sh

Execute using: bash testif2.sh

```
#!/bin/bash
if [ 1 -eq 1 ];
then
    echo "1 is equal to 1"
else
    echo "The numbers are not equal"
fi

1 is equal to 1
```

3. Check if a file exists

```
#!/bin/bash
if [ -f data.txt ]
then
   echo "File exists"
else
   touch data.txt
   echo "New file created"
fi
```

```
> then
>     echo "File exists"
> else
>     touch data.txt
>     echo "New file created"
> fi
New file created
```

4. String comparison

```
#!/bin/bash
user="admin"
if [ "$user" = "administrator" ];
then
    echo "The string matches"
else
    echo "The strings do not match"
fi
```

```
> then
>     echo "The string matches"
> else
>     echo "The strings do not match"
> fi
The strings do not match
```

5. Authentication check

```
#!/bin/bash
auth_users="admin"
read -p "What's your name? " user
if [[ "$user" == "$auth_users" ]];
then
    echo "You are an administrator"
elif [[ -z "$user" ]];
then
    echo "Please enter a username"
    read user
    echo "Hello $user, Greetings!"
else
    echo "You are a standard user"
fi
```

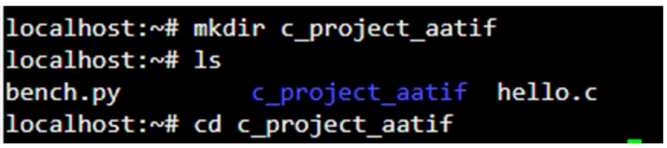
Experiment 07

Aim: Building C Project using makefile.

A Makefile is a script used by the make command to automate the process of compiling and linking programs. It defines a set of rules that specify how to build a project efficiently.

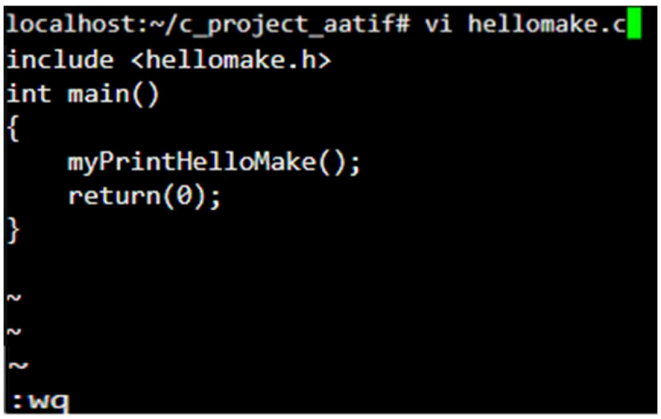
1. Main Project Directory

```
mkdir c_project_aatif
ls
cd c_project_aatif
```



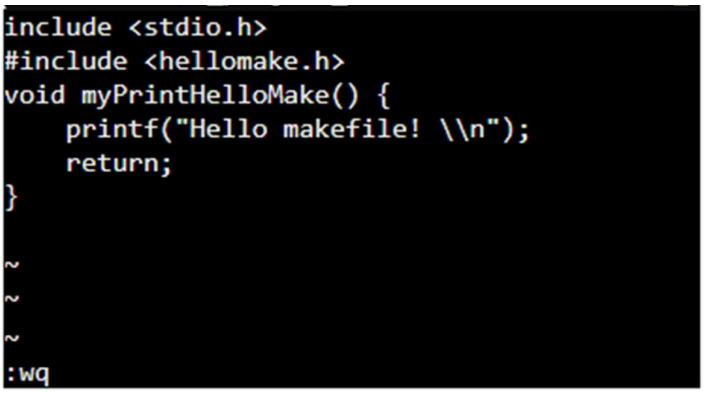
2. Create hellomake.c

```
vi hellomake.c
#include <hellomake.h>
int main()
{
    myPrintHelloMake();
    return(0);
}
 esc (for escaping insert mode and
come back to command mode)
:wq(for save and exit)
```



3. Create hellofunc.c

```
vi hellofunc.c
#include <stdio.h>
#include <hellomake.h>
void myPrintHelloMake() {
    printf("Hello makefile! \\n");
    return;
}
esc (for escaping insert mode and
come back to command mode)
:wq(for save and exit)
```



4.Create Makefile

vi makefile
CC=clang
CFLAG=-I.
DEPS = hellomake.h

%.o: %.c $(DEPS)

$(CC) -c -o $@ $< $(CFLAG)

hellomake: hellomake.o hellofunc.o

$(CC) -o hellomake hellomake.o hellofunc.o

esc (for escaping insert mode and come back to command mode)
:wq(for save and exit)

```
localhost:~/c_project_aatif# vi makefile
CC=clang
CFLAG=-I.
DEPS = hellomake.h

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

hellomake: hellomake.o hellofunc.o
    $(CC) -o hellomake hellomake.o hellofunc.o

~
~
~
:wq
```

5.Create hellomake.h

vi hellomake.h
/* example include file */
void myPrintHelloMake();
esc (for escaping insert mode and come back to command mode)
:wq(for save and exit)

```
localhost:~/c_project_aatif# vi hellomake.h
/* example include file */
void myPrintHelloMake();

~
~
~
:wq
```

6. Compile & Run

To compile the program:
make
To run the program:
./ hellomake