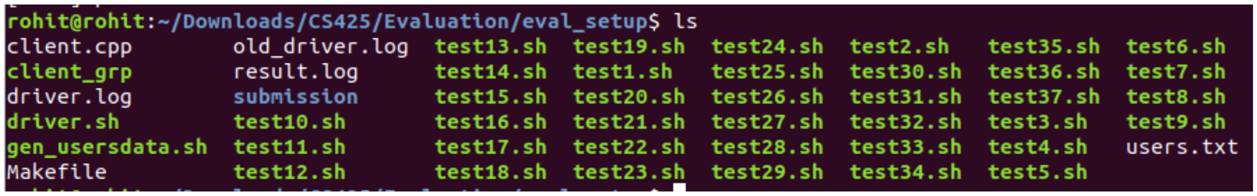
Q1) How to use the released evaluation setup?

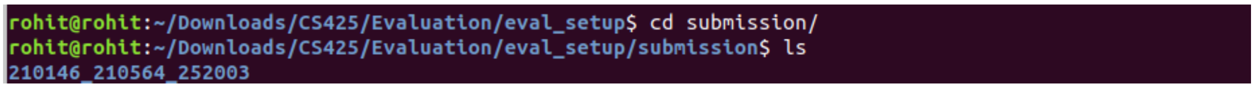
Ans:

1) Unzip the eval_setup.zip. It will create a ‘eval_setup’ directory with the following contents:



```
rohit@rohit:~/Downloads/CS425/Evaluation/eval_setup$ ls
client.cpp        old_driver.log  test13.sh  test19.sh  test24.sh  test2.sh   test35.sh  test6.sh
client_grp        result.log      test14.sh  test1.sh   test25.sh  test30.sh  test36.sh  test7.sh
driver.log        submission      test15.sh  test20.sh  test26.sh  test31.sh  test37.sh  test8.sh
driver.sh         test10.sh       test16.sh  test21.sh  test27.sh  test32.sh  test3.sh   test9.sh
gen_usersdata.sh  test11.sh       test17.sh  test22.sh  test28.sh  test33.sh  test4.sh   users.txt
Makefile          test12.sh       test18.sh  test23.sh  test29.sh  test34.sh  test5.sh
```

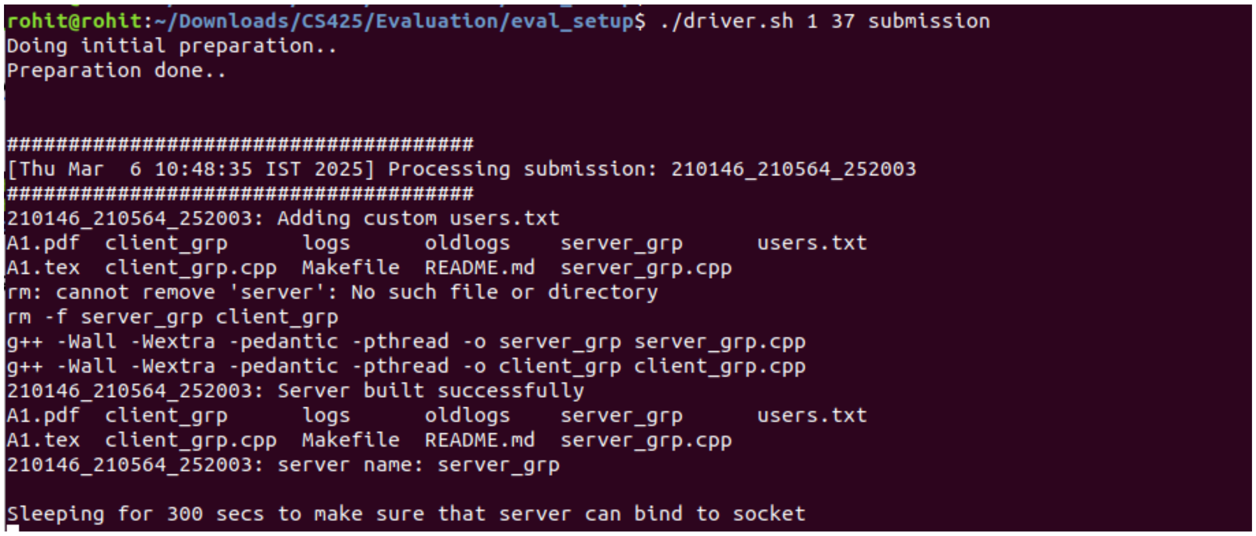
2) Add directory containing your submitted code to the ‘submission’ directory present in ‘eval_setup’.



```
rohit@rohit:~/Downloads/CS425/Evaluation/eval_setup$ cd submission/
rohit@rohit:~/Downloads/CS425/Evaluation/eval_setup/submission$ ls
210146_210564_252003
```

3) Run the driver script (driver.sh) present in the ‘eval_setup’ directory with the following command to trigger the evaluation:

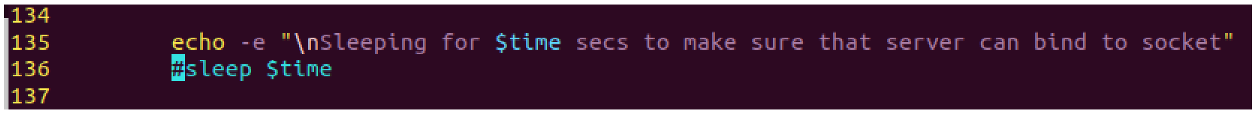
```
$ ./driver.sh     1     37     submission
```



```
rohit@rohit:~/Downloads/CS425/Evaluation/eval_setup$ ./driver.sh 1 37 submission
Doing initial preparation..
Preparation done..


#####################################
[Thu Mar  6 10:48:35 IST 2025] Processing submission: 210146_210564_252003
#####################################
210146_210564_252003: Adding custom users.txt
A1.pdf  client_grp      logs      oldlogs    server_grp      users.txt
A1.tex  client_grp.cpp  Makefile  README.md  server_grp.cpp
rm: cannot remove 'server': No such file or directory
rm -f server_grp client_grp
g++ -Wall -Wextra -pedantic -pthread -o server_grp server_grp.cpp
g++ -Wall -Wextra -pedantic -pthread -o client_grp client_grp.cpp
210146_210564_252003: Server built successfully
A1.pdf  client_grp      logs      oldlogs    server_grp      users.txt
A1.tex  client_grp.cpp  Makefile  README.md  server_grp.cpp
210146_210564_252003: server name: server_grp

Sleeping for 300 secs to make sure that server can bind to socket
```

Note:

a) Driver script sleeps for 300 secs before starting evaluation. You can avoid this by commenting out line number 136 in driver.sh by adding ‘#’ in front of the line.



```
134
135         echo -e "\nSleeping for $time secs to make sure that server can bind to socket"
136         #sleep $time
137
```

b) Driver script assumes that the name of your server program file is ‘server_grp.cpp’ (unless you are using a custom Makefile) or ‘server.py’
c) Driver script can take around ~ 1 hour to finish

Q2. How many test cases are used in evaluation?

Ans: 37

Q3. Can brief summary of testcases be provided?
Ans:

Basic Tests:
1) Valid login detected
2) Invalid login detected
3) Sending message to non-logged in user. Server should throw error.
4) Sending message to a logged in user.
5) Sending broadcast message to multiple users
6) Create group
7) Create group with same name as an already existent group
8) Join existing group
9) Join non-existing group
10) Send message to existing group
11) Send message to non-existant group
12) Leave group
13) Send message to group after leaving it

Basic Tests ++ :
14) Message user that terminated and then reconnected:
    a) Send msg to user u2
    b) u2 exits
    c) u3 joins
    d) u2 reconnects
    e) send msg to u2 again
15) Interleaving messages to multiple users
    a) Send msg to u2
    b) Send msg to u3
    c) Send msg to u4
16) Chain of msgs
    a) u1 sends msg to u2
    b) u2 to u3
    c) u3 to u4
    d) u4 to u1

17) Bidirectional communication
    a) u1 sends msg to u2
    b) u2 sends msg to u1
18) Broadcast while varying number of connected clients
    a) connect u1, u2, u3, u4
    b) broadcast by u1
    c) connect u5
    d) broadcast by u1
    e) disconnect u1
    f) broadcast by u2
19) Multiple clients can broadcast
    a) connect u1, u2, u3, u4
    b) u1 broadcasts
    c) u2 broadcasts
    d) u3 broadcasts
    e) u4 broadcasts
20) One user can create multiple groups
    a) u1 creates groups g1, g2, g3, g4
21) Multiple users can create groups
    a) u1 creates groups g1, g2
    b) u2 creates g3, g4
    c) u3 creates g5, g6
22) Users can join multiple groups
    a) u1 joins g1, g2, g3, g4
23) A group can have multiple members
    a) u2, u3, u4, u5 join g1
24) User sends multiple messages to a group without becoming its member
    a) u2 sends msg1, msg2, msg3, msg4 to g1
25) User can send multiple messages to same group after becoming its member
    a) u2 sends msg1, msg2, msg3, msg4 to g1
26) User can send message to multiple groups
    a) u5 sends msg to g1, g2, g3, g4
27) Multiple users can send messages to same group
    a) u2, u3, u4, u5 send msg to g1
28) User can leave multiple groups
    a) u5 leaves g1, g2
29) Multiple users can leave same group
    a) u2, u3, u4, u5 leaves g1
30) Multiple users can leave multiple groups
    a) u2,u3 leave g1
    b) u4,u5 leave g2

Inteleaved operations:

31) Private messages + broadcast

a) u1 sends private messages to u2,u3,u4
b) u2 broadcasts

32) Private messages + Group operations
   a) u2 sends private msg to u2,u3,u4
   b) u1 creates group
   c) u2,u3,u4 join group
   d) u2 sends message to group
   e) u2 leaves group

33) Broadcast + Group operations
   a) u1 broadcasts
   b) u1 creates group
   c) u2,u3,u4 join group
   d) u2 sends message to group
   e) u2 leaves group

34) Private messages + Group operations (This differs from test32 because we have changed order of operations)
   a) u1 creates group
   b) u2,u3,u4 join group
   c) send private msg to u2,u3,u4
   d) u2 sends message to group
   e) u2 leaves group

35) Broadcast + Group ops + Erroneous operation
   a) u1 creates group
   b) u2 broadcasts
   a) u3 joins non-existent group
   b) u2 broadcasts
   c) u3 join valid group
   d) u2 broadcasts

Little Stress:

36) Parallel group creation
   a) User u1 and u2 create 1000 groups each in parallel
   b) u1 joins all groups created by u2
   c) u2 joins all groups created by u1

37) Parallel login, logout
   a) 50 users login
   b) In parallel, 50 new users try to login while the existing 50 users try to logout
   c) Each user ‘j’ of the newly joined users sends a message to user ‘j+1’

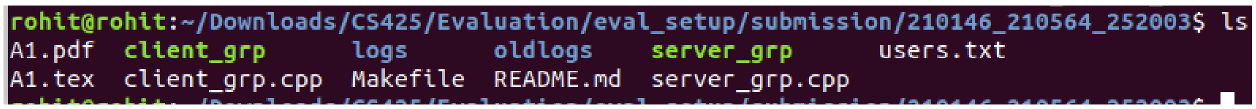
Q4. What is the grading scheme for testcases ?

Ans: (13 * 1) + (17 * 1.5) + (5 * 2.5) + (2 * 4.5) = 60

Q5. How to know the summary of evaluation i.e. which testcases passed and which failed? How to know the output generated by each client in each testcase?

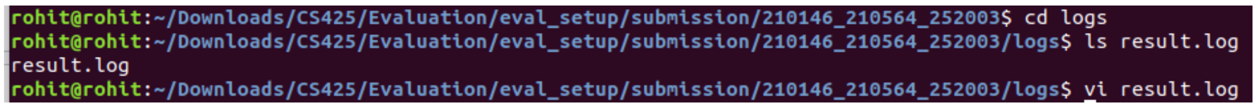
Ans:

1) Driver script creates log files in your code directory in a subdirectory named ‘logs’



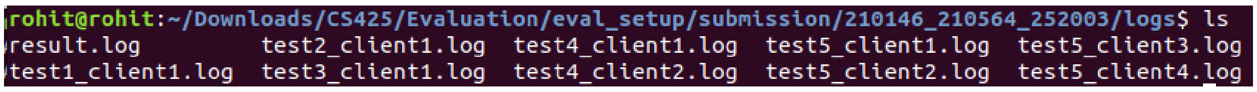
```
rohit@rohit:~/Downloads/CS425/Evaluation/eval_setup/submission/210146_210564_252003$ ls
A1.pdf   client_grp      logs      oldlogs     server_grp      users.txt
A1.tex   client_grp.cpp  Makefile  README.md   server_grp.cpp
```

2) File named ‘result.log’, present in ‘logs’ directory, stores the summary of the result



```
rohit@rohit:~/Downloads/CS425/Evaluation/eval_setup/submission/210146_210564_252003$ cd logs
rohit@rohit:~/Downloads/CS425/Evaluation/eval_setup/submission/210146_210564_252003/logs$ ls result.log
result.log
rohit@rohit:~/Downloads/CS425/Evaluation/eval_setup/submission/210146_210564_252003/logs$ vi result.log
```

3) Output produced by client ‘c’ in testcase ‘t’ is stored in a log file named ‘test’t’_client’c’.log’.   Eg: test4_client1.log

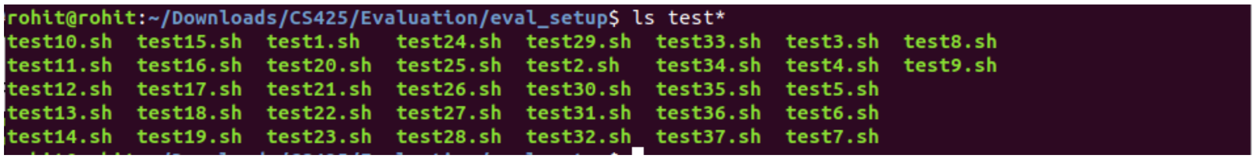


```
rohit@rohit:~/Downloads/CS425/Evaluation/eval_setup/submission/210146_210564_252003/logs$ ls
result.log         test2_client1.log  test4_client1.log  test5_client1.log  test5_client3.log
test1_client1.log  test3_client1.log  test4_client2.log  test5_client2.log  test5_client4.log
```

Q6. How can I see the testcase source code? How to understand the testcases source code?

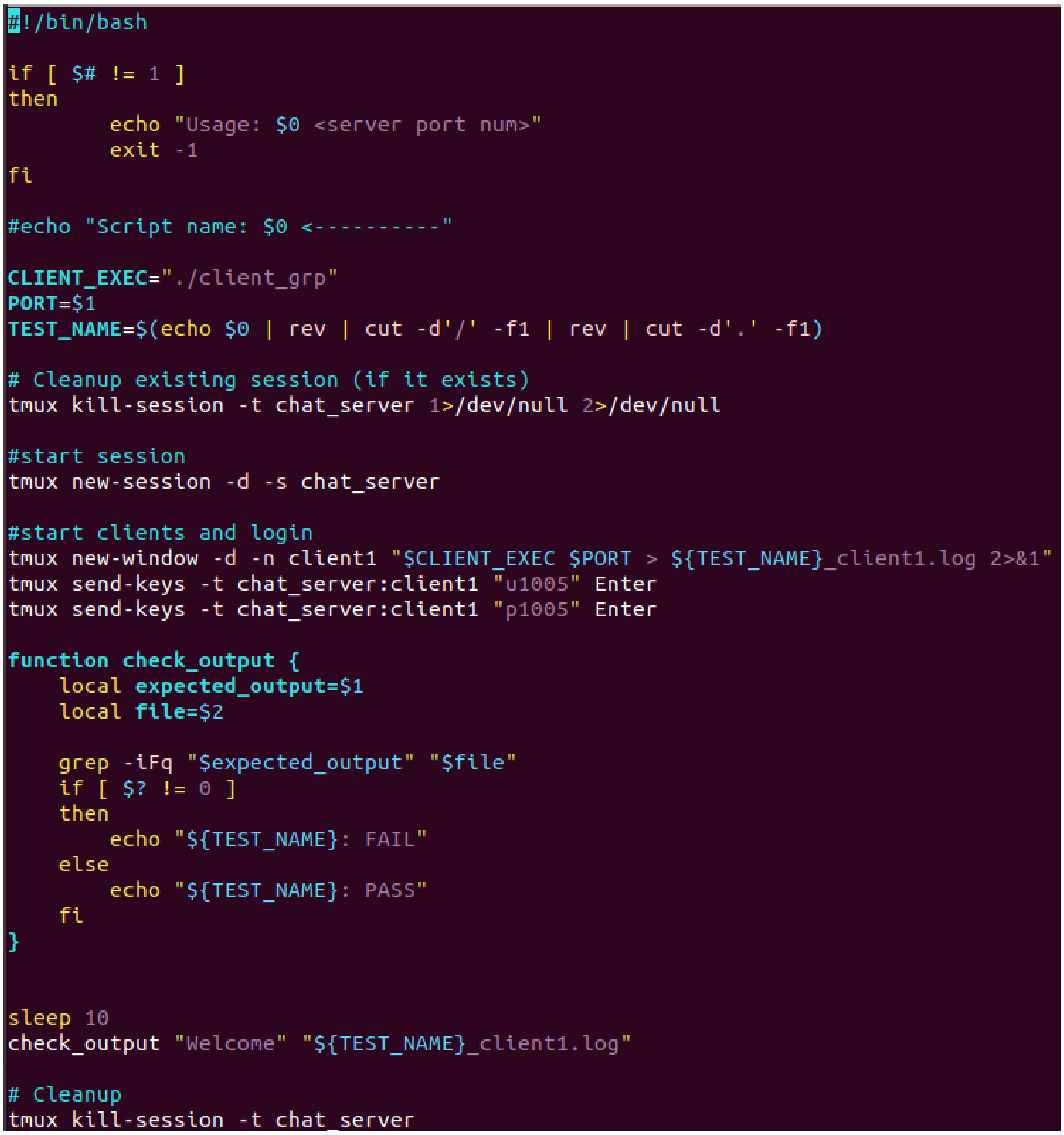
Ans:

- ‘test’t’.sh’ present in ‘eval_setup’ directory corresponds to testcase ‘t’
- Eg: test17.sh



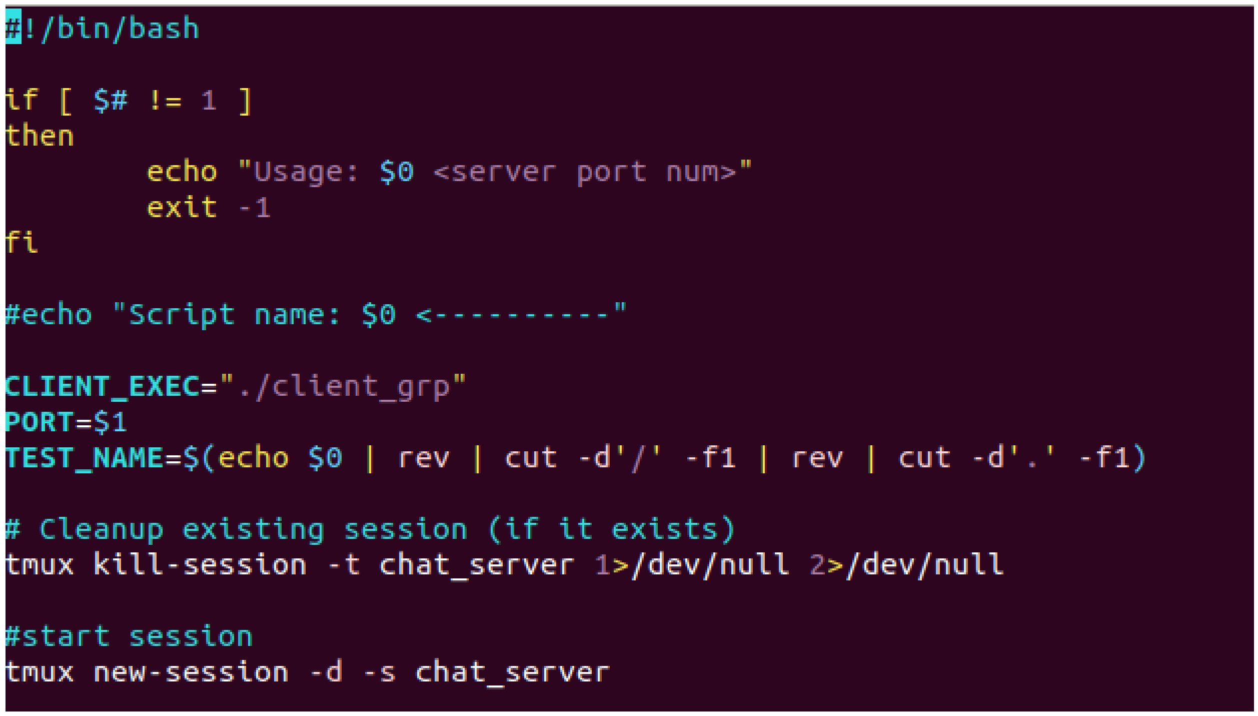
```
rohit@rohit:~/Downloads/CS425/Evaluation/eval_setup$ ls test*
test10.sh  test15.sh  test1.sh   test24.sh  test29.sh  test33.sh  test3.sh  test8.sh
test11.sh  test16.sh  test20.sh  test25.sh  test2.sh   test34.sh  test4.sh  test9.sh
test12.sh  test17.sh  test21.sh  test26.sh  test30.sh  test35.sh  test5.sh
test13.sh  test18.sh  test22.sh  test27.sh  test31.sh  test36.sh  test6.sh
test14.sh  test19.sh  test23.sh  test28.sh  test32.sh  test37.sh  test7.sh
```

- Let’s understand testcase 1



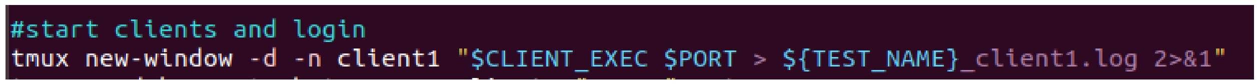
```bash
#!/bin/bash

if [ $# != 1 ]
then
        echo "Usage: $0 <server port num>"
        exit -1
fi

#echo "Script name: $0 <----------"

CLIENT_EXEC="./client_grp"
PORT=$1
TEST_NAME=$(echo $0 | rev | cut -d'/' -f1 | rev | cut -d'.' -f1)

# Cleanup existing session (if it exists)
tmux kill-session -t chat_server 1>/dev/null 2>/dev/null

#start session
tmux new-session -d -s chat_server

#start clients and login
tmux new-window -d -n client1 "$CLIENT_EXEC $PORT > ${TEST_NAME}_client1.log 2>&1"
tmux send-keys -t chat_server:client1 "u1005" Enter
tmux send-keys -t chat_server:client1 "p1005" Enter

function check_output {
    local expected_output=$1
    local file=$2

    grep -iFq "$expected_output" "$file"
    if [ $? != 0 ]
    then
        echo "${TEST_NAME}: FAIL"
    else
        echo "${TEST_NAME}: PASS"
    fi
}


sleep 10
check_output "Welcome" "${TEST_NAME}_client1.log"

# Cleanup
tmux kill-session -t chat_server
```

- Ignore initial code (shown below). It contains some basic initialisation and sanity checks.



```
#!/bin/bash

if [ $# != 1 ]
then
        echo "Usage: $0 <server port num>"
        exit -1
fi

#echo "Script name: $0 <----------"

CLIENT_EXEC="./client_grp"
PORT=$1
TEST_NAME=$(echo $0 | rev | cut -d'/' -f1 | rev | cut -d'.' -f1)

# Cleanup existing session (if it exists)
tmux kill-session -t chat_server 1>/dev/null 2>/dev/null

#start session
tmux new-session -d -s chat_server
```

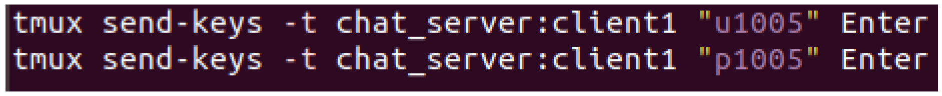
- Following code runs a client and redirects its output to ‘testcase’t’_client’c’.log’ file



```
#start clients and login
tmux new-window -d -n client1 "$CLIENT_EXEC $PORT > ${TEST_NAME}_client1.log 2>&1"
```

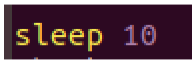
Note: Observe the string ‘client1’ after ‘-d -n’ in the above line of code. This is an identifier that will be used later to send data to the client. Each client will have a unique identifier like this.

- Input login details to the client. Observe, we have used the identifier ‘client1’ in the following commands to indicate that these login details should be entered as input to client1.
  Basically, we are simulating input to client1 i.e. as if a user typed “u1005” on keyboard and pressed ‘Enter’ in a terminal in which client1 was running.



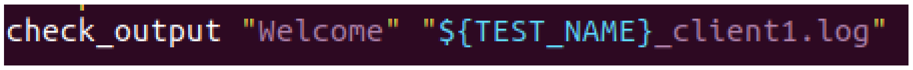
```
tmux send-keys -t chat_server:client1 "u1005" Enter
tmux send-keys -t chat_server:client1 "p1005" Enter
```

- We have entered login details on the client window. This must have triggered communication between client and server. Sleep for sometime to let this communication happen.



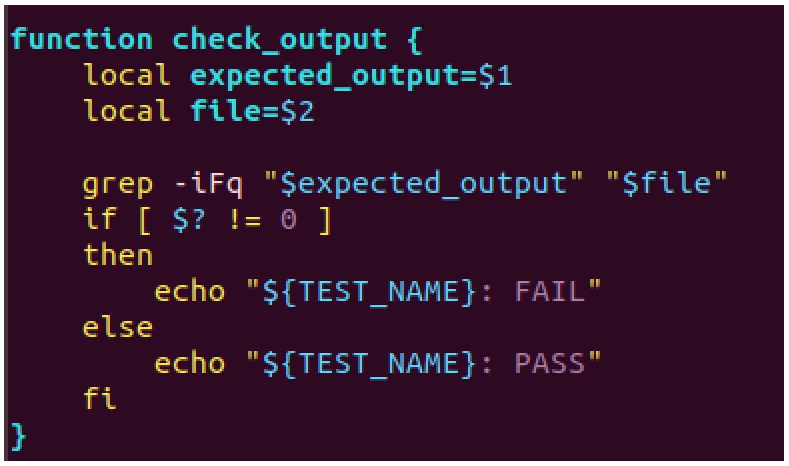
```
sleep 10
```

- Call a function named ‘check_output’. We are passing a string “Welcome” and name of the log file that contains the output of client1 to this function.



```
check_output "Welcome" "${TEST_NAME}_client1.log"
```

- ‘check_output’ function checks whether the string “Welcome” is present in the log file or not. If present, testcase is considered passed, else, failed.



```
function check_output {
    local expected_output=$1
    local file=$2

    grep -iFq "$expected_output" "$file"
    if [ $? != 0 ]
    then
        echo "${TEST_NAME}: FAIL"
    else
        echo "${TEST_NAME}: PASS"
    fi
}
```

- Other testcases follow similar logic

Q7. How many marks did I get for code quality?
Ans: Almost everyone got 15/15

Q8. How many marks did I get for Readme?
Ans: Following post (https://piazza.com/class/m5h01uph1h12eb/post/45) contained instructions on how the readme should look like.

- You got 25/25 if your readme is close to the told format
- You got 15/25 marks if it is not as per the provided instructions
- You got 5/25 marks if you did not put effort, just wrote couple of para’s
- You got 0/25 marks if readme is missing or empty

Q9. I got 0 or low marks. How can that happen?
Ans. One of the following things happened:

a) You didn't mention your group member's roll number's in your readme file.
We used automated scripts and without your roll number info, scripts had no way to know whom to assign marks.
(or)
b) Your submission didn't compile
(or)
c) Your server is buggy
(or)
d) Your server produced output in a format different from what was told (https://piazza.com/class/m5h01uph1h12eb/post/44)

**Q10. I was expecting 100/100 marks. But some testcases failed. Why?
More testcases are passing on my machine. I should get more marks.**

Ans. Most of the submissions had the following error:

- You have assumed that the recv() returns after receiving a single command from a client.
- However, this is not true. recv() simply returns when some data has been received from the client i.e. it is possible that recv() returns after receiving few bytes and it is also possible that recv() returns after receiving multiple messages
- Eg:
  i) Assume, client 1 sends '/msg john Hello' and '/msg bob Hi' to the server.
  ii) Now, it is possible that on server side, when recv() returns, its buffer is as following:
  [/msg john Hello /msg bob Hi]
  iii) You are checking only the prefix of the contents of the buffer and based upon this, you will deduce that the buffer contents are as following:

```
/msg                 <------ command
john                 <------ receiver
Hello /msg bob Hi  <------ message
```

  iv) As, a result, you will forward this message to john and bob will never receive the message.
  v) Hence, testcase fails.

- However, this error may not always be reproducible.
- Behaviour of your server on our test machine during evaluation will be considered final in this case.