CS 35L Winter 2019 Final Exam Solutions

Question 1.

1a) Creating hard link and soft (symbolic) links

1b) find . -maxdepth 2 -type l | wc -l

1c) baz = 3165230 , qux = cannot be determined from the data , will not be 3165232

1d) mv does not affect hard link. Hence baz will still have the same contents as before. Same as that of foo.

mv changes the soft link , hence qux will have a broken link and error while opening it (no contents)

1e) rm does not affect hard link. Hence baz will still have the same contents as before. Same as that of foo.

rm changes the soft link , hence qux will have a broken link and error while opening it (no contents)

1f) foo will be appended with the new contents of baz.

bar does not have write permissions as can be seen from the question screenshot. Hence nothing can be written to qux as well. echo "Updating files" >> qux will give error.

Question 2

2a. A. This will work because backticks are used.

B. This will not work, because single quotes do not preserve the meanings of the back ticks.

C. This will work, because double quotes preserve the meanings of the back ticks.

2b.

```
for FILE in "$LIST"
 do if [ ! -r "$FILE" ]
then
 echo $FILE
fi done
```

2c.

Using a basic regular expression will work because we search for the "-r" suffix which indicates a read permission.

An extended regular expression will also work.

2d.

You can assume that apart from the first letter of the firstname and lastname, the rest of the firstname and lastname is in lower case.

```
`^[A-Z][a-z]+[RCSQ][a-z]+ \. ENSG[0-9]+\([0-9]{2,}\)`
```

Question 3:

3a1) The Makefile does not contain a target named “all”. Hence when make all is typed on the terminal, it results in an error. make move on the other hand would have worked fine.

3a2)

i) In large projects, time-consuming re-compiles are avoided by makefiles.

ii) Maintaining dependencies across different files in a project becomes less cumbersome with the help of makefiles.

3b) patch -p3 < ../exam_patch.patch

3c)

```
def numDigits(n):
      return len(str(n))

def isArmstrong(n):
      if n < 0:
            raise ValueError("Please input a
non-negative integer!")
      numDig = numDigits(n)
      tot = 0
      for i in str(n):
            tot += int(i) ** numDig
      if tot == n:
            return "Yes"
      else:
            return "No"

n = int(input())
print("Is", n, "an armstrong number?",
isArmstrong(n))
```

Question 4:

4a. the

4b. _dog

4c. jumped

4d. over_the

4e. _brown_fox

Question 5:

5a)

Yes, the program will execute. The output is:

fd1=3, fd2 = 4

Called read which returned 12

Content read: Charlie_

5b) content1.txt = Lord_of_the_Charlie_

content2.txt = Charlie_Chaplin

5c) content1.txt = Lord_of_the_Rings

Charlie_

5d) content2.txt = Charlie_Lord_of_the_

5e) Open -> fopen()

Close -> fclose()

Read -> fread()

Write -> fwrite()

5f) Expecting an answer on the lines of buffered and unbuffered I/O

5g)

```
int main(){
int fd1,fd2,sz1,sz2;
char *c1 = (char *) calloc(100, sizeof(char));
char *c2 = (char *) calloc(100, sizeof(char));
fd1 = open(“content1.txt”, O_RDWR);
fd2 = open(“content2.txt”, O_RDWR);
if (fd1 < 0){
perror(“Error in opening content1.txt”);
exit(1);
}
```

```
if (fd2 < 0){
perror("Error in opening content2.txt");
exit(1);
}

sz1 = read(fd1, c1, 20);
sz2 = read(fd2, c2, 20);
write(1, c1, strlen(c1));
write(1, c2, strlen(c2));
close(fd1);
close(fd2);
}
```

Question 6.
Omitted as it's on threading.

Question 7.
7a.

```
all:
criu
criu: main.o foo.o bar.o
 gcc -o criu main.o foo.o bar.o

main.o: main.c foo.h bar.h type.h
 gcc -o main.o main.c

foo.o: foo.c
 gcc -o foo.o foo.c

bar.o: bar.c
 gcc -o bar.o bar.c

clean:
 rm ./*.o

tarball:
 tar -cjvf clean.tar.gz main.c foo.c bar.c
bar.h type.h foo.h MakeFile
```

7b.

```
protection:
chmod 444 ./*.c
chmod 444 ./*.h
make criu
chmod 744 ./*.c
chmod 744 ./*.h
```

Question 8.
8a) iv 8b) i 8c) ii 8d) i 8e) iv

Question 9
i. C
ii.

```
Application.py
dbconn.py*
frontend.py*
index.html
README.md
structure.css
format.py+
test_data.csv+

*: modified
+: added
```

iii.

```
git clone
git checkout iss42 -b "(any branch name)"
```

iv.

```
`git log | grep -B 10 "issue49.*testcases"`
```

(-B 10 gives the ten lines before the match, you would've got extra credit for using this flag, but no points lost for not using it)
OR `git log > git-log.txt`

open git-log.txt in

e-macs, use extended regexp search (C-M-s) to look for "issue49.*testcases" and locate the commit

Fall 2017 Zhaowei Tan Question #2

```
#!/bin/bash

fake_path='path1:path2:path3'

poly () {
  echo "$1" ' ' "$2"
  result=$(($1*$1*$1+$2*$2+$1*$2))
  echo 'The result is' "$result"
}

if [ ! "$1" ] || [ "$2" ]; then
  echo 'Only 1 argument allowed'
  exit 1
fi

if [ "$1" ]; then
  if [ ! -d "$1" ] || [ -L "$1" ]; then
    echo 'Argument is not a valid directory'
    exit 1
  fi
  directory="$1"
fi

echo 'directory is' "$directory"

path2=`echo $fake_path | tr ":" " "`
# OR: path2=${fake_path//:/ }
echo "$path2"
in_path=0
for a_path in $path2
do
  echo "$a_path"
  if [ $a_path = $directory ]; then
     echo 'match found'
     in_path=1
  fi
done
```

```
if [ "$in_path" == 0 ]; then
  echo 'match NOT found'
  fake_path+=:"$directory"
fi

file=`ls -prt | grep -v / | head -n 1`
x=${#file}
# y=0
# all_files=`ls`
# for a_file in $all_files
# do
#  if [ ${#a_file} -eq 2 ]; then
#    y=$((y+1))
#  fi
#done

#alternative method, one-liner
y=`ls | awk 'length == 2' | wc -l`

poly "$x" "$y"

echo 'fake path is' "$fake_path"
```