Solutions to CITS2003/CITS4407 In Semester Test 2022

Q1. Given the file name pattern `a?b[0-9]*` which of the following file names match that pattern **(2 marks each**):

`abb3`	Match	Does Not Match
`ab3`	Match	Does Not Match
`abb3.txt`	Match	Does Not Match
`abc3`	Match	Does Not Match

Q2. I executed the command: `file_count = $(ls $1| wc -l)` but got back the unexpected response: file_count: not found
Write the fixed command **(2 marks)**

```
Ans: file_count=$(ls $1| wc -l)    # no spaces around =
```

Q3. At the top of a Bash script, you will typically find:

```
#!/usr/bin/env bash
```

You might alternatively see

```
#!/bin/bash
```

Whichever of these you choose, why is that command placed there? **(2 marks)**

```
Ans: It allows the shell to interpret the script as being a
Bash script (versus Python, Shell, etc)
```

Q4. This question has several parts, but together they are the text of a Bash script, which is called `extract_lines`. As input `extract_lines` is given two integers, L1 and L2 (representing line numbers) and the name of a text file. It then reports on standard output the lines from L1 to L2, inclusive, e.g. `extract_lines 100 200 Alice_in_Wonderland.txt`.

The `extract_lines` script begins with:

```
#!/usr/bin/env bash
```

You then need to start with some anti-bugging tests.

Q4.1. Write a shell command which tests whether the command entered by the user has 3 arguments **(3 marks)**

```
Ans:
if [[ $# -ne 3 ]]
then
  echo "Usage: $0 <bottom line number> <top line number>
<textfile>" > /dev/stderr
  exit 1
fi
```

```
Any message will do, so long as there is one.
```

Q4.2. Write a Shell command which tests whether the third argument is an ordinary file with length greater than zero. **(3 marks)**

```
Ans:
if [[ ! -s $3 ]]
then
  echo "The file $3 does not exist or has zero lenth"
/dev/stderr
  exit 1
fi
```

Q4.3. Describe one further antibugging test that should be done. **(2 marks)**

```
Ans: you could test whether second integer is greater than or
equal to the first integer
```

Q4.4. Now that processing the call to `extract_lines` has survived the anti-bugging, write the shell commands which implement reporting the range of lines. There are lots of ways of doing this; one way is to loop over the lines of the file using a `for` loop that looks like:

```
IFS="     # Set the end of word marker to get whole text lines
"
for line in $(< file)
do

done
```

What are the missing Shell commands? (Hint: you will likely need a line counter) **(10 marks)**

```
Ans:
counter=0
for i in $(< $3)
do
  counter=$((counter + 1))
  if [[ $counter -ge $1 ]]
  then
    echo $counter $i
    if [[ $counter -eq $2 ]]
    then
      exit 0
    fi
  fi
done
```

Q4.5. Another way of doing the Q4.4 computation (after antibugging) can make use of built-in Unix programs plus, perhaps, one or two other shell commands, but not Shell loops. What is the code for that that? **(6 marks)**

Ans:

```
diff=$(( $2 - $1 + 1))
head -n $2 $3 | tail -n $diff
```

Q5. I want to create the shell script `nth`, which given an integer N and a file of numbers, returns the Nth largest number, so `nth 1 F` should return the largest value in F, `nth 2 F` the second largest value, etc. Don't worry if there are more than one equally large values, just return one of them. If you wish you can refer to the `extract_lines` program that you defined in Q4. **(4 marks)**

Ans:

```
sort -k 1nr $2 > _x
extract_lines $1 $1 _x
```