BUSA8090 Assignment 1

Byeong Jun Park – 42808146

Q1

File name foo, goo and hoo file has been created via following codes

touch -t 202103081800 foo

touch -t 202103081801 goo

touch -t 202103081802 hoo

As we can see below, we can see that the files were successfully created with foo being the oldest, goo next & hoo the earliest.

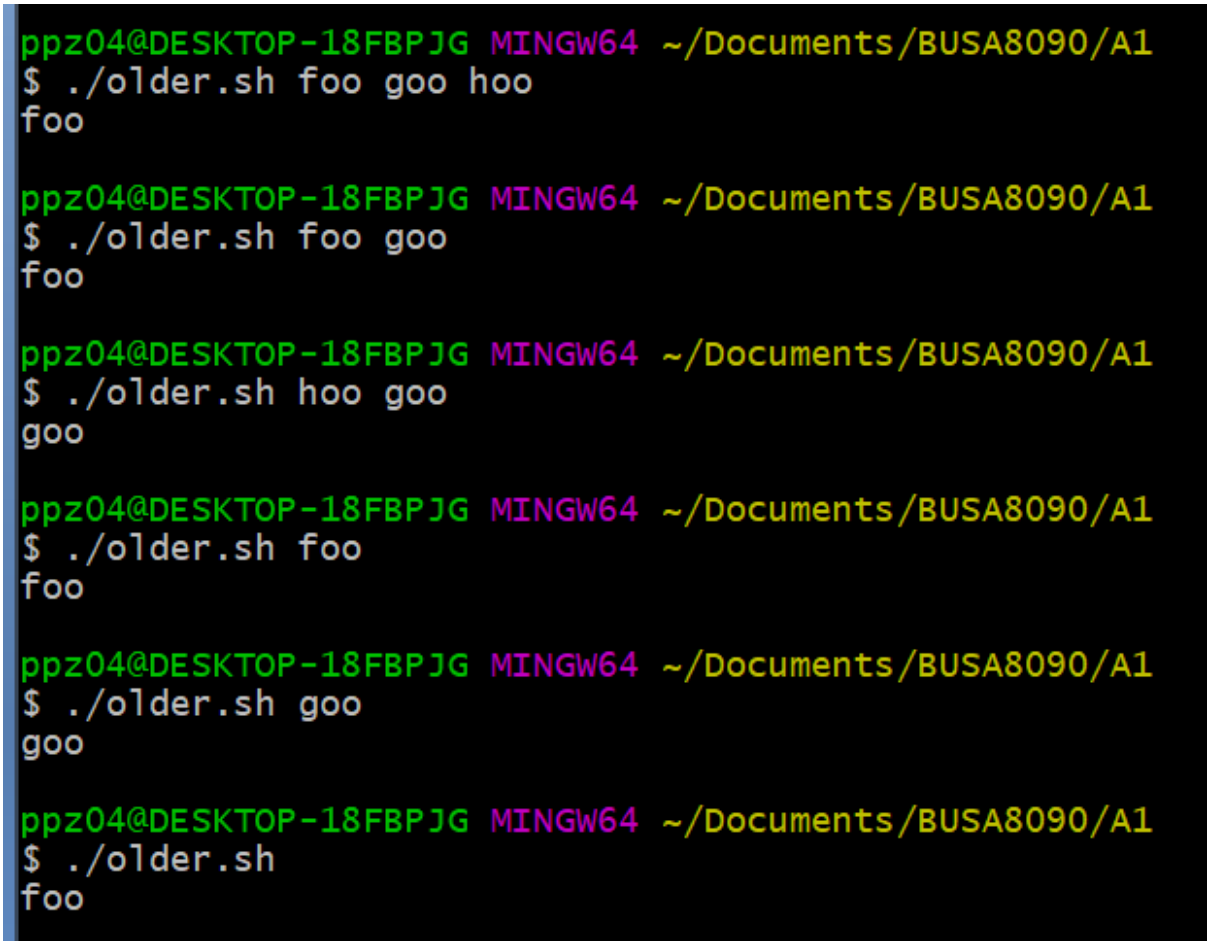
```
ppz04@DESKTOP-18FBPJG MINGW64 ~/Documents/BUSA8090/A1
$ ls -lt
total 5
-rw-r--r-- 1 ppz04 197609 320 Mar 17 22:05 chim.sh
-rwxr-xr-x 1 ppz04 197609 321 Mar 17 19:55 case-cp.sh*
-rwxr-xr-x 1 ppz04 197609 356 Mar 17 19:36 funny.sh*
-rwxr-xr-x 1 ppz04 197609 272 Mar 17 19:15 older.sh*
-rwxr-xr-x 1 ppz04 197609 271 Mar 17 19:13 trial.sh*
-rw-r--r-- 1 ppz04 197609   0 Mar  8 18:02 hoo
-rw-r--r-- 1 ppz04 197609   0 Mar  8 18:01 goo
-rw-r--r-- 1 ppz04 197609   0 Mar  8 18:00 foo
```

older.sh script was created with the following codes.

```
#!/bin/bash

ls $* -1 -t  | tail -1

#ls isa list command
#$* are the provided arguments. For this use case, the provided file.
#-1 option list one file per line
#-t sort by amendment date, newest to oldest

#tail -1 gives the last observation, which would be the oldest
```

As explained in the commentaries above, (ls $*) lists the provided arguments, (-1) option to list one file name per line, (-t) to sort the arguments by newest to oldest then (tail – 1) to extract the last line. In summary, the script sorts the provided names of the files (arguments) from newest to oldest then picks the last one to spit the name of the oldest file.

As we can see below, the script is successful for all combinations.



When 3 files were invoked into the argument, it provided the oldest file name foo.

When 2 files, hoo and goo, were invoked, it provided the older file.

When only 1 file is invoked, it only provides that file as that is still the oldest.

When no argument is invoked, it provides the oldest file in the directory.

Q2

The following codes were created to create funny.sh

```
#!/bin/sh
if [ -z  "$*" ]; then echo "This is NOT funny"
else  echo "This is funny"
fi



#-[-z "$*"] means if the  arguement, "$*", is empty, -z "%*"  outputs
#True , condition is satisfied, "This is Not funny" is written.

#If the argument is not empty, in which -z "%*" will give false, it goes
#into else condition and print out "This is Not funny".
```

if statement is used to check if an argument was included or not included what the script is invoked. –z checks whether something is empty or not and "%*" refers to argument.

So in the if condition, -z "%*" will return True if the argument is not invoked (i.e. argument is empty). If the condition is True, it satisfies the if condition and then it will print "This is Not funny".

If the argument is invoked, argument is not empty, –z "%*" returns False, it goes into else phase and print "This is funny".

```
ppz04@DESKTOP-18FBPJG MINGW64 ~/Documents/BUSA8090/A1
$ ./funny.sh
This is NOT funny

ppz04@DESKTOP-18FBPJG MINGW64 ~/Documents/BUSA8090/A1
$ ./funny.sh asfasdf
This is funny

ppz04@DESKTOP-18FBPJG MINGW64 ~/Documents/BUSA8090/A1
$ ./funny.sh asfasdfasdf asdfsd
This is funny
```

As we can see above, the script is successful. When an argument is not included, 'This is Not funny' is printed. When argument/arguments are invoked, 'This is funny' is printed.

Q3

a) case-cp.sh is a public file located in the following url.

https://www.staff.hs-mittweida.de/~wuenschi/data/media/compbiolbook/chapter-10-shell-programming--case-cp.sh

we can use curl URL > filename to download the file to the directory.

```
ppz04@DESKTOP-18FBPJG MINGW64 ~/Documents/BUSA8090/A1
$ curl https://www.staff.hs-mittweida.de/~wuenschi/data/media/compbiolbook/chapter-10-shell-programming--case-cp.sh > case-cp.sh

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   321  100   321    0     0    127      0  0:00:02  0:00:02 --:--:--   127
```

b) The following script was used to answer the question.

```
#<!/bin/sh

Minutes=$(date +%M)
count=0

if [[ $Minutes -ge 0 && $Minutes -lt 20 ]]; then  exit
elif [[ $Minutes -ge 20 && $Minutes -lt 40 ]]; then echo  -e "\a"
else    while test $count -lt 2; do
        echo -e "\a"
        sleep 1
        count=$[$count+1]
        done
fi
```

Minutes=$(date +%M) Extracts the minutes of the local time and saves it as “Minutes”. This was used to write the future codes easier.

count=0 was initially created to create loop easier.

If [[$Minutes –ge 0 && $Minutes –lt 20]]; means if the extracted minutes is greater than equal to 0 and less than 20, both must satisfy, then we simply exit the script and nothing happens.

If the above condition is not true, we go into the next if condition below.

[[$Minutes –ge 20 && $Minutes –lt 40]]; means if the extracted minutes is greater than equal to 20 and less than 40, both must satisfy, then it presents the beep signal generated by the escape sequence “\a”.

IF the above condition is also not true, we go into the else condition.

While test $count –lt 2; -> This means while the variable count is less than 2 loop is repeated. The variable count was initially set as 0. At the very start, count is 0 <2, condition is satisfied and the loop begins. It provides the beep sound via echo –e “\a” then rests one second via “sleep 1”. Count is then redefined as count +1. So after the first loop, count now becomes 1. Since count is still less than 2, it goes into the loop again producing a beep sound, resting one second then adding additional to make count 2. Count is now greater than 2, loop is exited.

Since minutes can only range from 0 to 59, anything that doesn’t satisfy the previous conditions would all fall minutes from 40 to 59 and satisfy the conditions defined in the question.