OPERATING SYSTEMS
SHELL PROGRAMMING 2

1. RECAP

- bash script example:

```
#!/bin/bash

pwd
ls
```

Important:

- to run a bash script we must add execution permissions first, then run with ./scriptname:

```
chmod +x script_1.sh
./script_1.sh
```

- comments start with `#(hash);` `#!/bin/bash` is a special (required) comment
- test, expr, `evaluate command ` , $(evaluate command), $((eval arithm expr)), [test], [[pattern matching regex]], ((arithm expression)), ${! param expansion}, $#{length of this var string}
- *To do a simple numeric comparison (or any other shell arithmetic), use (( )) instead of test. To test variables you should quote the "variablename" as they may undergo word splitting or globbing, with New test [[this is not necessary (read more: https://ss64.com/bash/test.html)*

```
  – [ "$DEMO" = 5 ]  #numerical comparison
  – [[ $DEMO == 10 ]]
  – [[ $a == z* ]]   # True if $a starts with an "z" (wildcard pattern matching).
  – [[ $a == "z*" ]] # True if $a is equal to z* (literal matching).
  – [ $a == z* ]     # File globbing and word splitting take place.
  – [ "$a" == "z*" ] # True if $a is equal to z* (literal matching).
  – [[ $s =~ [^0-9]+([0-9]+) ]]   # Regex matching (direct)
  – pat='[^0-9]+([0-9]+)'  #use variables for regex matching
  – s='I am a string with some digits 1024'
  – [[ $s =~ $pat ]] # $pat must be unquoted
  – echo "${BASH_REMATCH[0]}" # The captured groups (match results) are available in an array named BASH_REMATCH. The 0th index in the BASH_REMATCH array is the total match
  – echo "${BASH_REMATCH[1]}"  #The i'th index in the BASH_REMATCH array is the i'th captured group, where i = 1, 2, 3 ...; here it will print "1024"
  – if [[ "$a" < "$b" ]] # ASCII alphabetical order
  – if [ "$a" \< "$b" ]  # ASCII alphabetical order
  – if [ -z "$s" ]   # String is null
```

- more information: man bash or man command

2. SOLVED PROBLEMS

1. Write a bash script that counts all the C files from a given directory and all of its subdirectories.

```
#!/bin/bash


if [ $# -lt 1 ]; then
    echo "Insufficient arguments"
    exit 1
fi


find $1 -type f | grep -E -c "\.c$"
```

1a. Write a bash script that counts all the lines of code in the C files from the directory given as command-line argument, excluding lines that are empty or contain only spaces.

```
#!/bin/bash
if [ -z "$1" ]; then
    echo "No parameters given"
    exit 1
fi
if [ ! -d "$1" ]; then
    echo "Parameter is not a folder"
    exit 1
fi
total=0
for f in $(ls "$1" | grep -E "\.c$"); do
  if test -f "$1/$f"; then
    nr_lines=$(grep -E -c -v "^[ \t]*$" "$1/$f")
    echo "$f: $nr_lines"
    total=$((total+nr_lines))
  fi
done
echo "Total lines: $total"
```

1b. Write a bash script that counts all the lines of code in the C files from the directory given as command-line argument and all its subdirectories, excluding lines that are empty or contain only spaces.

```
#!/bin/bash
if [ -z "$1" ]; then
    echo "No parameters given"
    exit 1
fi
if [ ! -d "$1" ]; then
    echo "Parameter is not a folder"
    exit 1
fi
total=0
for f in $(find "$1" -type f | grep -E "\.c$"); do
  nr_lines=$(grep -E -c -v "^[ \t]*$" $f)
  echo "$f - $nr_lines"
  total=$((total+nr_lines))
done
echo "Total lines: $total"
```

2. Write a bash script that receives any number of command line arguments and prints on the screen, for each argument, if it is a file, a directory, a number or something else.

```
#!/bin/bash
while [ ! $# -eq 0 ]; do
  arg=$1
  if test -f $arg; then
    echo "$arg is a regular file"
  elif [ -d $arg ]; then
    echo "$arg is a directory"
  elif echo $arg | grep -E -q "^[0-9]+$"; then
    # the regular expression here can be adapted to match any type of number, as needed
    echo "$arg is an integer number"
  else
    echo "$arg is something else"
  fi
  shift
done
```

3. Write a bash script that keeps reading strings from the keyboard until the name of a readable regular file is given.

```
#!/bin/bash

fname=""

while [ ! -f "$fname" ]; do

  read -p "Enter a string: " fname

done
```

4. Write a bash script that sorts the file given as command line arguments in ascending order according to their file size in bytes.

```
#!/bin/bash

for f in $@; do

  if test -f $f; then

    du -b $f

  fi

done | sort -n
```

5. Write a script that receives as command line arguments pairs consisting of a filename and a word. For each pair, check if the given word appears at least 3 times in the file and print a corresponding message.

```
#!/bin/bash

if [ $# -lt 2 ]; then
    echo "Please provide at least 2 arguments"
    exit 1
fi

if [ $(($# % 2)) -eq 1 ]; then
    echo "You must provide an even number of arguments"
    exit 1
fi

while [ $# -gt 1 ]; do
    file=$1
    word=$2

    if [ ! -f "$file" ]; then
        echo "Name $file is not a file"
    else
```

```
        count=$(grep -E -o "\<$word\>" "$file" | wc -l)
        if [ $count -ge 3 ]; then
            echo "Word $word appears $count times in file $file"
        fi
    fi
    shift 2
done

if [ $# -eq 1 ]; then
    echo "Warning: final pair is incomplete"
fi
```

6. Find recursively in a given directory all the symbolic links, and report those that point to files/directories that no longer exist. Use option -L to test if a path is a symbolic link, and option -e to test if it exists (will return false if the target to which the link points does not exist)

```
#!/bin/bash


for link in $(find "$1" -type l); do
    if [ ! -e "$link" ]; then
        echo "Link $link is not valid"
    fi
done
```

7. Write a bash script that receives a folder name as argument. Find recursively in the folder the number of times each file name is repeated.

```
#!/bin/bash

if [ -z "$1" ]; then
    echo "Please provide one argument"
    exit 1
fi

if [ ! -d "$1" ]; then
    echo "Argument must be a directory"
    exit 1
fi

find "$1" -type f | awk -F/ '{print $NF}' | sort | uniq -c
```

8. Write a script that receives program/process names as command line arguments. The script will monitor all the processes in the system, and whenever a program with one of those names is run, the script will kill it and display a message. (see commands ps, kill, killall). Alternativ, comenzile pkill/pgrep pot fi folosite.

```
#!/bin/bash

if [ $# -eq 0 ]; then
    echo "Provide at least one name"
    exit 1
fi

while true; do
    for process in $@; do
        PIDs=""
        PIDs=$(ps -ef | awk '{print $8" "$2}' | grep -E "\<$process " | awk '{print $2}')
        if [ -n "$PIDs" ]; then
            kill -9 $PIDs
        fi
    done
    sleep 3
done
```

9. Consider a file containing a username on each line. Generate a comma-separated string with email addresses of the users that exist. The email address will be obtained by appending "@scs.ubbcluj.ro" at the end of each username. Make sure the generated string does NOT end in a comma.

```
#!/bin/bash
if [ -z "$1" ]; then
    echo "Please provide one input file"
    exit 1
fi

if [ ! -f "$1" ]; then
    echo "The given argument is not a file"
    exit 1
fi

result=""
for u in $(cat "$1"); do
    result="$u@scs.ubbcluj.ro,$result"
done
```

```
result=$(echo $result | sed -E "s/,$//")

echo $result
```

10. Create a bash script that finds all the text files in a specified folder (the current folder if there is no specified folder). For all such files, the script will report the filesize, permissions and number of unique lines.

```
#!/bin/bash

dir=${1:-"."}

if [ -d "$dir" ]; then
    for f in $(find "$dir" -type f); do
        if file $f | grep -E -q "text"; then
            size=$(du -b $f | cut -f1)
            perm=$(ls -l $f | cut -d' ' -f1)
            lines=$(sort $f | uniq | wc -l)
            echo "Filename: $f - size: $size - permissions: $perm - unique lines: $lines"
        fi
    done
fi
```

3. EXERCISES – Shell

1. Write a script that reads filenames and check for each file how many words contains on the first line and the size of the file. Perform all required validations on the input data.

(commands: grep, wc, head)

2. Same as 1, but input by command line arguments.

3. Write a shell script that receives as argument a natural number N and generate N text files:

- the name of the files will be of the form: file_X, where X={1,2,…, N}

- each generated file will contain online lines between X and X+5 of the file /etc/passwd

(commands: touch, sed/awk)

4. Calculate the average of all process ids in the system per user.

(commands: ps)

5. Write a shell script that for each command line parameter will do:

- if it’s a file, print the name, number of characters and lines in the file

- if it’s a directory, print the name and how many files it contains (including subdirectories files)

(commands: test, wc, awk, find)

6. Write a shell script that received triplets of command line arguments consisting a filename, a word and a number (validate input data). For each such triplet, print all the lines in the filename that contain the word exactly k times.

(commands: shift, awk)

7. Write a shell script that, for all the users in /etc/passwd, creates a file with the same name as the username and writes in it all the ip addresses from which that user has logged in. (hint: use the last command to find the ip addresses)

8. Write a script that finds recursively in the current folder and displays all the regular files that have write permissions for everybody (owner, group, other). Then the script removes the write permissions from everybody. Hint: use chmod's symbolic permissions mode (see the manual).

9. Create a bash script that displays every second the process count per user sorted descending by process count for all users specified as command line arguments. If no arguments are given, the script will display the process count per user for all users. Validate usernames provided.

10. Write a shell script that receives any number of words as command line arguments, and continuously reads from the keyboard one file name at a time. The program ends when all words received as parameters have been found at least once across the given files.