Task 1. bash script named `script1`

The name of your bash script must be **script1** and below is a description of what it should do when executed.

1. The script displays `Outer_Directory created` and creates in the current directory a subdirectory named `Outer_Directory` .
2. In `Outer_Diretcory` it creates 15 ASCII text files named `proj1_file`, ..., `proj15_file` . The content of `proj1_file` consists of two lines: the first saying `generated value=100` and the second saying `Next file is proj2_file with generated value=200` . The content of `proj2_file` consists of two lines `generated value=200` and `Next file is proj3_file with value=300` etc. Note that the last file `proj15_file` contains two lines, the first saying `generated value=1500` and the second saying `Next file does not exist` . Also note that the value is always 100 times the index (*this part ought to be done mostly by a loop, not by 15 separate commands*).
3. Then the script displays `contents of the created files` and for each created file it displays its name followed by a colon on one line and then all the lines of the file (*this part ought to be done mostly by a loop, not by 15 separate commands*).
4. Then the script displays `Inner_Directory created` and creates in `Outer_Directory` a subdirectory named `Inner_Directory` .
5. Then it moves all files created in step 2 that contain character `3` or `6` to `Inner_Directory` (*this should be done in a loop*).
6. Then the script displays `Outer_Directory regular files` and displays names of regular files in the directory. (*Note that you cannot use* `ls` *as it displays all entities in the directory and not just regular files*)
7. Then the script displays `Inner_Directory regular files` and displays names of regular files in the directory.
8. Then the script concatenates all files moved to `Inner_Directory` in step 5 into a file named `EVERYTHING` and deletes these files.
9. Then it displays all files in `Inner_Directory` .
10. Then the script displays a message `Inner_Directory and all its files removed` and removes all files from `Inner_Directory` and when empty, it removes the directory `Inner_directory` using the `rmdir`
11. Then the script displays `Outer_Directory and all its files removed` and removes all files from `Outer_Directory` and when empty, it removes the directory `Outer_directory` using the `rmdir` .
12. Then it displays `Current Directory` and displays all files in the current directory. The current directory now should contain exactly the same files and subdirectories as just before this script was executed.

What commands you might need: `cd mkdir echo ls grep mv cat` and ***for loop***. *Note, that after the script has been executed, the current directory is in the same state as it was when the*

script started its execution. Please, try emulate as close as possible the format of the display of the sample run below.

A few useful hints:

- *Current directory is referred to as* `.` *, the parent directory as* `..`
  For instance, `ls .` *will show all files/subdirectories in the current directory, while* `ls ..` *will show all files/subdirectories in the parent directory*
- *a range from 1 to 15 can be expressed as* `{1..15}`*, for instance* `for i in {1..15}`
- *to concatenate a string with a number* (`x` *contains a string,* `i` *contains a number*), *use* `$x$i`
- *to concatenate a number with a literal string* (`i` *contains a number*), *use* `$i"world"`. If the value of `i` is 2, it will produce a string `2world` .
- *to increment a variable* `i` *containing a number, use* `i = $(($i+1))`
- *to multiply a variable* `i` *containing a number and store the result in* `x`*, use* `x = $((100*$i))`
- *to test if a name stored in a variable* `x` *is a name of a regular file in current directory, use* `[ -f $x ]`
- *Another way to deal with regular files is to use command* `find . -type f` *, it will output a list of names of all the regular files in the current directory, a you can check whether the name is in the list.*
- *to figure out if a file whose name is stored in a variable* `x` *contains a symbol* `2` *or*`7`*, use* `y = `grep '2\|7' $x`` *and then test whether* `y` *is empty* (*note the* `'` *quotes inside and the* `` ` `` *quotes around the whole expression*).
- *If you create by mistake a file called* `-XXX` *where* `XXX` *stands for any name, you must remove it using command* `rm -- -XXX` *. The usual command* `rm -XXX` *will not work as* `rm` *would think that* `-XXX` *is a switch.*
- *In bash you quite often need to check to see if a variable has been set or has a value other than an empty string. This can be done using the* `-n` *or* `-z` *string comparison operators. The* `-n` *operator checks whether the string is not null. Effectively, this will return true for every case except where the string contains no characters. ie*:

  ```
  VAR="hello"
  if [ -n "$VAR" ]
  then
      echo "VAR is not empty"
  fi
  ```

  Similarly, the `-z` *operator checks whether the string is null. ie*:

  ```
  VAR=""
  if [ -z "$VAR" ]
  then
  ```

```
        echo "VAR is empty"
    fi
```

Note the spaces around the square brackets. Bash will complain if the spaces are not there.

A sample run:

```
Outer_Directory created
Created files
proj1_file:
generated value=100
Next file is proj2_file with generated value=200
proj2_file:
generated value=200
Next file is proj3_file with generated value=300
proj3_file:
generated value=300
Next file is proj4_file with generated value=400
proj4_file:
generated value=400
Next file is proj5_file with generated value=500
proj5_file:
generated value=500
Next file is proj6_file with generated value=600
proj6_file:
generated value=600
Next file is proj7_file with generated value=700
proj7_file:
generated value=700
Next file is proj8_file with generated value=800
proj8_file:
generated value=800
Next file is proj9_file with generated value=900
proj9_file:
generated value=900
Next file is proj10_file with generated value=1000
proj10_file:
generated value=1000
Next file is proj11_file with generated value=1100
proj11_file:
generated value=1100
Next file is proj12_file with generated value=1200
proj12_file:
generated value=1200
Next file is proj13_file with generated value=1300
proj13_file:
generated value=1300
Next file is proj14_file with generated value=1400
```

```
proj14_file:
generated value=1400
Next file is proj15_file with generated value=1500
proj15_file:
generated value=1500
Next file does not exist
Inner_Directory created
Outer_Directory regular files
proj10_file
proj11_file
proj14_file
proj15_file
proj1_file
proj4_file
proj7_file
proj8_file
proj9_file
Inner_Directory regular files
proj12_file
proj13_file
proj2_file
proj3_file
proj5_file
proj6_file
EVERYTHING
proj12_file
proj13_file
proj2_file
proj3_file
proj5_file
proj6_file
Inner_Directory and all its files removed
Outer_Directory and all its files removed
Current Directory
script1
script2
```

Task 2. bash script named `script2`

The name of your bash script must be **script2** and below is a description of what it should do when executed.

1. First the script **script2** checks the command line arguments. It should have 1 or 2 or 3 command line arguments (*we are not counting the name of the script, thus `script2 a` is considered to have one command line argument. We abbreviate coomand line argument as CLA*).

If 3 CLAs are used, they must be `-0 <file1> <file2>`
If 2 CLAs are used, they must be `-1 <file>`
If 1 CLA is used, it must be `-2`

2. If the number of CLAs is wrong, the script displays an error message `wrong number of command line arguments` , followed by what we call ***usage***

```
3. Usage: script2 -0 <file1> <file2>
4.     or
5.        script2 -1 <file>
6.     or
          script2 -2
```

and terminates.

7. If 3 CLAs are used and the first CLA is not `-0` , the script displays an error message `incorrect command line argument: X` followed by ***usage*** and terminates; where `X` is the value of the first CLA. If the first CLA is `-0` , then the script displays a message `creating file X` where `X` is the value of the second CLA (i.e. `<file1>`), then it creates a file named `X` that contains just 1 li ne `test1`, and then it displays the contents of the file using `cat`. Then the script displays a message `creating file Y` where `Y` is the value of the third CLA (i.e. `<file2>`), then it creates a file named `Y` that contains just 1 line `test2`, and then it displays the contents of the file using `cat`. Then the script exits. The two created files `X` and `Y` must be left intact in the current directory.
8. If 2 CLAs are used and the first CLA is not `-1` , the script displays an error message `incorrect command line argument: X` followed by ***usage*** and terminates; where `X` is the value of the first CLA. If the first CLA equals `-1`, the script displays: `testing file X` where `X` is the value of the second CLA (i.e. `<file>`) and terminates.
9. If 1 command line argument is used and it does not equal `-2`, the script displays an error message `incorrect command line argument: X` followed by ***usage*** and terminates, where `X` is the value of the first command line argument. If the first command line argument equals `-2` , then the script displays `Good bye` and terminates.

A few useful hints:

- *The number of command line arguments is stored in* `$#` *variable. Play with it to determine whether it counts CLAs as we do* (*without the name of the script*), *or if it includes the name of the script in the count* (*i.e., whether it includes in the count* `$0`)
- *Since* `$0` *is a pathname, we can extract the name of the script using* `basename`, *e.g.* `basename $0`
- *To terminate execution of a script, you can use the* `exit` *command.*

Sample runs: executing `script2 a b c d`

```
incorrect number of command line arguments
Usage: script2 -0 <file1> <file2>
    or
       script2 -1 <file>
    or
       script2 -2
```

Sample runs: executing `script2 a b`

```
incorrect command line argument: a
Usage: script2 -0 <file1> <file2>
    or
       script2 -1 <file>
    or
       script2 -2
```

Sample runs: executing `script2 -1 hello`

```
testing file hello
```

Sample runs: executing `script2 -2 hello`

```
incorrect command line argument: -2
Usage: script2 -0 <file1> <file2>
    or
       script2 -1 <file>
    or
       script2 -2
```

Sample runs: executing `script2`

```
incorrect number of command line arguments
Usage: script2 -0 <file1> <file2>
    or
       script2 -1 <file>
    or
       script2 -2
```

Sample runs: executing `script2 -2`

```
Good bye
```