CS 392, Systems Programming: Assignment 1

1. **Objective**

   Your task is to write a simple bash script to provide the basic functionality of a recycle bin. In addition to moving files into the recycle bin directory, the script must be able to list and purge the files that have been placed into the recycle bin. This script acts as a substitute for the `rm` command, giving the user a chance to recover files deleted accidentally. Note that restoring the files is not part of this exercise.

   You should consult with the following websites for details about programming in bash:
   http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html
   http://tldp.org/LDP/abs/html/

2. **Problem**

   You will create a bash script called `junk.sh` that operates exactly as follows. To ensure the test cases found in this document make sense, create a directory called `junk` inside the home directory of the cs392 user in your lubuntu VM. Create your `junk.sh` script inside the `junk` directory and run it from there.

   a. When the script is executed, it first parses the command line arguments. The usage message is shown below to help you determine what features you need to implement.

```
$ ./junk.sh
Usage: junk.sh [-hlp] [list of files]
   -h: Display help.
   -l: List junked files.
   -p: Purge all files.
   [list of files] with no other arguments to junk those files.
```

   Notice there are four flags [h, l, p] that the script must handle. You must getopts in your solution to receive credit for this part of the assignment. When no argument is supplied or when the user passes -h, the script should produce the output seen in the box above.

   If an unexpected flag is found, the script should display an error message and repeat the correct usage. This is true, even if the user inputs too many flags. If one is unknown, the script terminates with the message seen below.

```
$ ./junk.sh -z
Error: Unknown option '-z'.
Usage: junk.sh [-hlp] [list of files]
   -h: Display help.
   -l: List junked files.
   -p: Purge all files.
   [list of files] with no other arguments to junk those files.
```

If more than one (valid) flag is specified, the script should display an error message and repeat the correct usage. See below.

```
$ ./junk.sh -l -p
Error: Too many options enabled.
Usage: junk.sh [-hlp] [list of files]
   -h: Display help.
   -l: List junked files.
   -p: Purge all files.
   [list of files] with no other arguments to junk those files.
```

If one or more flags are specified and files are supplied, the script also tell the user that too many options have been supplied.

```
$ ./junk.sh -l note.txt
Error: Too many options enabled.
Usage: junk.sh [-hlp] [list of files]
   -h: Display help.
   -l: List junked files.
   -p: Purge all files.
   [list of files] with no other arguments to junk those files.
```

b. After parsing the command line arguments, the script checks for the presence of the `~/.junk` directory. Note, `.junk` is a hidden folder placed in the **home directory** of the user. If the directory is not found, the script creates it.
c. At this point, the script can assume that is ready to do its main task. So, depending on what flag (if any) the user supplied, the script needs to either display the usage message, list the files in the recycle bin, purge the files in the recycle bin, or move the files or folders specified on the command line into the recycle bin.

   If a file or folder is not found, the junk script should warn the user. If the user is attempting to junk multiple files or folders and some of them are not found, the script should warn about the missing ones and actually move forward with copying the ones that exist to the `.junk` directory. See below for the expected output.

```
$ ./junk.sh notfound1.txt found.txt notfound2
Warning: 'notfound1.txt' not found.
Warning: 'notfound2' not found.
```

d. When listing files in the recycle bin, use `ls -lAF`.
e. Be sure to exit the script with a 0 for success and 1 for an error, as other processes might need to know the return value of this script before proceeding.

3. Sample Run Time Scenario

Below is the output expected when running this script on several common scenarios. Your output must match this output verbatim.

```
$ pwd
/home/cs392/junk
```

```
$ touch junk0.txt
$ mkdir -p dir1
$ mkdir -p dir2/dir3
$ mkdir .hideme
$ touch dir1/junk1.txt
$ touch dir2/junk2.txt
$ touch dir2/dir3/junk3.txt
$ tree
.
├── dir1
│   └── junk1.txt
├── dir2
│   ├── dir3
│   │   └── junk3.txt
│   └── junk2.txt
├── junk0.txt
└── junk.sh

3 directories, 5 files

$ ./junk.sh junk0.txt

$ ./junk.sh -l
total 0
-rw-rw-r-- 1 cs392 cs392 0 Feb  3 17:50 junk0.txt
<Of course, the date and time will almost certainly be different.>

$ ./junk.sh dir1/junk1.txt

$ ./junk.sh -l
total 0
-rw-rw-r-- 1 cs392 cs392 0 Feb  3 17:50 junk0.txt
-rw-rw-r-- 1 cs392 cs392 0 Feb  3 17:50 junk1.txt

$ ./junk.sh dir2/dir3/junk3.txt

$ ./junk.sh .hideme

$ ./junk.sh -l
total 4
drwxrwxr-x 2 cs392 cs392 4096 Feb  3 17:50 .hideme/
-rw-rw-r-- 1 cs392 cs392    0 Feb  3 17:50 junk0.txt
-rw-rw-r-- 1 cs392 cs392    0 Feb  3 17:50 junk1.txt
-rw-rw-r-- 1 cs392 cs392    0 Feb  3 17:50 junk3.txt

$ tree
.
├── dir1
├── dir2
│   ├── dir3
│   └── junk2.txt
└── junk.sh

3 directories, 2 files

$ tree -a ~/.junk
/home/cs392/.junk
├── .hideme
├── junk0.txt
├── junk1.txt
```

```
└── junk3.txt

1 directory, 3 files

$ ./junk.sh -p

$ ./junk.sh -l
total 0

$ tree -a ~/.junk
/home/cs392/.junk

0 directories, 0 files
```

---

4. Requirements

You **MUST** use:

a) **getopts** to parse command line arguments
b) a **here document** for constructing the help message
c) the **basename** utility inside the here document to get the script name without additional directory information
d) the **readonly** keyword when setting up the variable name for the `.junk` directory at the top of the script