CS 392: Homework Assignment 1
Due: February 2, 11:55pm

Philippos Mordohai
Department of Computer Science
Stevens Institute of Technology
Philippos.Mordohai@stevens.edu

Collaboration Policy. Homeworks will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems. Use of the Internet is allowed, but should not include searching for previous solutions or answers to the specific questions of the assignment.

Late Policy. No late submissions will be allowed without consent from the instructor. If urgent or unusual circumstances prevent you from submitting a homework assignment in time, please e-mail me explaining the situation.

Objective

Your task is to write a 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 in 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 may consult http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html and http://tldp.org/LDP/abs/html/ for details about programming in bash, and https://www.stackchief.com/tutorials/Bash%20Tutorial%3A%20getopts specifically for `getopts`.

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 `test` inside the home directory of the user in your VM. Create your `junk.sh` script inside the `test` directory and run it from there. In the following examples, we assume the username is `user`; however, you should not hard code this into your code (otherwise you will fail almost every test case).

Use `getopts` to Parse Arguments

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 three 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 also true, when 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 tells 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.
```

Performing Tasks

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.

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 attempts 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.
```

When listing files in the recycle bin, use `ls -lAF`.

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. You can see the return value of a process using `echo $?`

Sample Run Time Scenario

Below is the output expected when running this script on several common scenarios. Your output must match this output verbatim, except username, date/time, and empty lines.

```
$ pwd
/home/user/test
$ 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 user user 0 Feb 3 17:50 junk0.txt

$ ./junk.sh dir1/junk1.txt

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

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

$ ./junk.sh .hideme

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

$ tree
.
```

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

3 directories, 2 files

$ tree -a ~/.junk
/home/user/.junk
├── .hideme
├── junk0.txt
├── junk1.txt
└── junk3.txt

1 directory, 3 files


$ ./junk.sh -p
$ ./junk.sh -l
total 0

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

0 directories, 0 files
```

Requirements

You MUST use:

(1) `getopts` to parse command line arguments;

(2) a here document for constructing the help message;

(3) the `basename` utility inside the here document to get the script name without additional directory information;

(4) the readonly keyword when setting up the variable name for the `.junk` directory at the top of the script.

Note that your code has to be **portable** – the directory of `.junk` should be set up correctly on any machine and any user.

Hints

- Your output must match the example **verbatim**. Examples include but not limited to:
  - The last four lines in the usage message must start with exactly **three** spaces;
  - The symbols, capitalizations, *etc* must match the sample;
  - The order of the flags must be the same;
  - ...
- The only exception of matching the sample output is the line starting with `Usage`. In the sample output, `junk.sh` is the name of the script, so the output shows its name in usage message. However, you must not hard-code this. During testing, your script will be renamed to something else, and your output must reflect this change. A hard-coded filename will fail most of the test cases;
- Your `getopts must not print system generated errors;`
- `Whenever you echo an error message, it must be printed through standard error only. During the test, if the standard error stream does not detect output, or the standard out stream does detect extra output, you will fail the test case;`
- `You must exit 1 when there's an error (`*`e.g.,`*` illegal options); However, only printing the help message does not count as error.`

Deliverable

Only submit the `.sh` file through Canvas.