CSE 15L Scripting Project: KillByName

Due Dates:
Part 1: Monday, November 21, 2016 at 11:59 pm, PST
Part 2: Wednesday, November 30, 2016 at 11:00 am, PST

Grade Breakdown:

Correctness/Execution:	9%
Commenting/Style:	1%
Total:	10% of your course grade

Part 1 and Part 2 are worth 30% and 70% respectively of the overall score.

How to Get Started:

Login to your CSE 15L account and do the following: (cs15x???)

```
$ mkdir ~/KillByName
$ cp ~/../public/KillByName/KillByName.empty ~/KillByName
$ cd ~/KillByName
```

Assignment Overview:
For this assignment, you will be working on a BASH script, whose purpose is to kill *all* processes that satisfy the following conditions:

1. Belongs to your User ID (UID)
2. Matches either the given Process ID (PID), or the given executable name
3. If the process is a parent process, kill all its children first before killing the parent.

You may pass in two option flags to configure how the script behaves:

1. **The graceful kill option (-g):** If the user runs your script with the '-g' flag, your script should only attempt to kill processes with **`kill`** and not **`kill -9`**. This means that if a process traps signal 15 (the default signal sent by **`kill`**), that process should not be killed immediately. On the other hand, if the '-g' flag is not present, your script should first attempt to kill processes using just **`kill`**, and if they are still alive after **`kill`**, then you should proceed to use **`kill -9`**

2. **The name option (-n):** The default behavior of your script takes a PID as argument and kills the process that matches with that PID. If the user wishes to kill all processes associated with a name, the '-n' flag should be passed to the script, followed by the name as its argument.

NOTICE: You may only use the **`kill`** command when terminating processes. You may **not** use any other commands that will let you look up processes based on their name such as **`pkill`**, **`killall`**,**`pgrep`**, ***etc***.

Part 1 (30%):

Your task for Part 1 is to write a function in your script called **`usage()`** that checks whether the script is used correctly.

The correct usage should be:

```
$  KillByName [-g] -n <name> or KillByName [-g] <pid>
```

Your script should detect an illegal usage of your script's options on the command line. For example, if a user typed:

```
$  ./KillByName -n
$  ./KillByName -g
```

Your script should print out an error message explaining the appropriate usage of your script.

If there are no processes by the given name, your script should NOT print out any kind of error message. Instead it should exit with an exit status of 1. Otherwise, if the specified processes are successfully killed, then your script should exit with an exit status of 0.

Additionally, the **`usage()`** function will check what flags are, or not, present. If the flags are invalid, or missing arguments, you must display the **`$USAGE**.

Once **`usage()`** is complete, you need to check to make sure that the process name that is passed into the 'n' flag exists.

In order to write this functionality, you will need to use the **`getopts`** functionality. **Using if-statements will not be sufficient.**

Here are some outputs that we expect for Part 1:

```
$  ./KillByName -g 24601
graceful kill is present

$  ./KillByName -g
Usage: KillByName [-g] -n <name> or KillByName [-g] <pid>
```

```
$  ./KillByName -g -n
Usage: KillByName [-g] -n <name> or KillByName [-g] <pid>

$  ./KillByName -g -n bash
graceful kill is present
process name is present

$  ./KillByName -gn bash
graceful kill is present
process name is present

$  ./KillByName -n bash -g
graceful kill is present
process name is present
```

~~~ END OF PART 1 ~~~

Part 2 (70%):

For the second part, you will need to be able to kill the process name or pid that is passed into the script.

Keep in mind that each child process must be killed before its parent. In this assignment, you only need to kill direct child processes. **You do not need to worry about grandchildrens or beyond**. We recommend to do this by storing all children processes in an array through a loop, and kill them separately in another loop.

We wrote a utility program for you to test your script. You can easily see what processes owned by you are running by using:

```
$  check
```

To create processes, you can open a new terminal, and simply just open any file.
To create children processes, you can open do the following:

```
$  vim foo &
$  vim bar &
$  vim baz &
$  check
USER       PID  PPID COMMAND
cs15x     2793  2793 -bash
cs15x     1121  2793  \_ vim foo
cs15x     1123  2793  \_ vim bar
cs15x     1125  2793  \_ vim baz
```
~~~

```
cs15x      1195   2793   \_ check
```

Notice that bash’s process id is 2793, and it’s children’s processes also share 2793 as their PPID, but their personal PID is unique.

~~~ END OF PART 2 ~~~

~~~