George Suarez
Lab 2
20 points total

Lab 2 – Basic Shell Programming

Output of *ginfo*:

```
Hello georgesuarez
Today is
Mon Apr  9 18:27:45 PDT 2018
Number of user login :
     2
Calendar
    April 2018
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8     10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

What difference do you see when executing the script with

```
$ ./ginfo
```

and with

```
$ . ./ginfo
```

Why?

- When executing *./ginfo*, it just executes the script then awaits for the next command. When executing . *./ginfo*, it is sourcing the content of *ginfo* into the current shell.

Variables in Shell

- Current working directory:

```
$ echo $PWD
/Users/georgesuarez
```

- Login name:

```
$ echo $USER
georgesuarez
```

- Shell Version:

```
$ echo $BASH_VERSION
3.2.57(1)-release
```

- Home Directory:

```
$ echo $HOME
/Users/georgesuarez
```

- Number of columns of current terminal:

```
$ echo $COLUMNS
80
```

User Defined Variables (UDV)

- How do you define variable *x* with value 10 and print it on screen?

```
georgesuarez at MacBook-Pro in ~
$ x=10
georgesuarez at MacBook-Pro in ~
$ echo $x
10
```

- How do you define variable *xn* with value 'Rani' and print it on screen?

```
georgesuarez at MacBook-Pro in ~
$ xn=Rani
georgesuarez at MacBook-Pro in ~
$ echo $xn
Rani
```

- How do you print the sum of two numbers, say, 6 and 3?

```
georgesuarez at MacBook-Pro in ~
$ echo `expr 6 + 3`
```

```
9
```

- How do you define two variables *x*=20, *y*=5 and then print the quotient of *x* and *y* (i.e *x/y*)?

```
georgesuarez at MacBook-Pro in ~
$ x=20
georgesuarez at MacBook-Pro in ~
$ y=5
georgesuarez at MacBook-Pro in ~
$ echo `expr $x / $y`
 4
```

- Modify the above question to store the result of dividing *x* by *y* to a variable called *z*.

```
georgesuarez at MacBook-Pro in ~
$ z=`expr $x / $y`
georgesuarez at MacBook-Pro in ~
$ echo $z
4
```

Executing *./testShell.sh* and *echo $XYZ*:

```
georgesuarez at MacBook-Pro in ~
$ ./testShell.sh && echo $XYZ
```

Outputs a blank line, meanwhile:

```
georgesuarez at MacBook-Pro in ~
$ . ./testShell.sh && echo $XYZ
```

Outputs the number 2017

The difference between is the two is that the first test is executing the script, but it does not source the contents into the current shell that is being executed in, so the shell does not know the variable XYZ outside of the script. Meanwhile, the second test does source the variable into the shell which means the shell knows the variable XYZ and it can output its content which in this case is the number 2017.

AWK

Executing:

```
georgesuarez at MacBook-Pro in ~
$ ps auxw | awk '{print $1 "\t\t" $2}'
```

Outputs a list of users and its associated process id. The command is executing *ps auxw* and then sending the output of that command by using the pipe operator to the *awk* command which handles what is going to be outputted which in this case it is printing the first field followed by two tabs then the second field.

Starting New Processes

Running the program *test_system.cpp* with the command *ps -ax* outputs all the process ids followed by its's TTY, duration of the process, and the command that is running on the computer.

```
//test_system.cpp
#include <stdlib.h>
#include <iostream>

using namespace std;

int main()
{
   cout << "Running [s with system\n";

      system("ps -ax ");

      cout << "Done \n";

      return 0;
}
```

Running the same program, but with the *ps -ax &* command shows the same output as before, but now the & makes the command run in the background in a subshell and awaits any additional commands since the return status is 0.

Shell Programming Practice

What does the option "-v" in the **grep** command do?

- The "-v" in the **grep** command selects the lines that are not matching with any of the specified patterns.

terminateProcess script:

```
for pid in $(ps -e -f | grep $1 | grep -v grep | grep -v $0 | awk '{print $2}')
do
   kill $pid
done

if [ "$pid" == "" ]
then
        echo "No such process exists"
fi
```

Testing *terminateProcess* script:

```
georgesuarez at MacBook-Pro in ~/University/cse460/Lab 2 on master*
$ ./robot &
[1] 30201
georgesuarez at MacBook-Pro in ~/University/cse460/Lab 2 on master*
$ ./robot &
[2] 30212
georgesuarez at MacBook-Pro in ~/University/cse460/Lab 2 on master*
$ ./robot &
[3] 30223
georgesuarez at MacBook-Pro in ~/University/cse460/Lab 2 on master*
$ ps -l
  UID   PID  PPID       F CPU PRI NI       SZ    RSS WCHAN     S          ADDR TTY          TIME CMD
  501 28095 28094    4006   0  31  0  4296240   1828 -       S             0 ttys000    0:00.67 -bash
  501 30201 28095    4006   0  31  0  4267724    780 -       R             0 ttys000    1:13.52 ./robot
  501 30212 28095    4006   0  31  0  4267724    788 -       R             0 ttys000    1:11.45 ./robot
  501 30223 28095    4006   0  31  0  4267724    780 -       R             0 ttys000    0:07.50 ./robot
georgesuarez at MacBook-Pro in ~/University/cse460/Lab 2 on master*
$ ./terminateProcess.sh robot
[1]   Terminated: 15         ./robot
[2]-  Terminated: 15         ./robot
[3]+  Terminated: 15         ./robot
georgesuarez at MacBook-Pro in ~/University/cse460/Lab 2 on master*
$ ./terminateProcess.sh robot
No such process exists
```

Discussion: I managed to finish all the sections in this lab with some difficulty in the last section of the lab. I had to research the syntax for the *if-else* and *for-loop* to get a better. After

figuring that out, then it was pretty simple what to do next which was to use the kill command on the *pid*, and then checking if the *pid* exists at all. Since I finished all the sections in this lab, I would give myself 20/20 points.