Department of Engineering Science

Interfacing with Raspberry Pi 3 Model B

Updated: 9/19/20

A. Objectives

1. Learn about ping utility and ICMP
2. Review netstat and SSH
3. Learn about ARP protocol
4. Basic package installation in Linux
5. Review shell scripting
6. Learn about basics of Python programming
7. Learn how to use Python and Shell scripts to control GPIO ports on the Pi
8. Combining shell scripting and Python

B. Time of Completion

This laboratory activity is designed for students with very little knowledge of Raspberry Pi and it is estimated to take about 3 hours to complete.

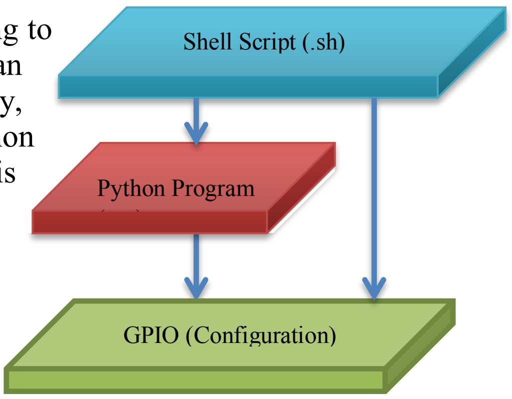
C. Requirements

1. A Raspberry Pi 3 Model 3
2. 32 GByte MicroSD card → Give your MicroSD card to the lab instructor for a copy of Ubuntu.
3. USB adaptor to power up the Pi

D. Pre-Lab

1. Make sue you have completed RPI lab 1.
2. Read about GOIP on your PI have: https://learn.sparkfun.com/tutorials/raspberry-gpio
3. Review about Python programing. http://www.learnpython.org/
4. Review netstat commands: https://www.tecmint.com/20-netstat-commands-for-linux-network-management/
5.

E. Lab



In this lab we learn how to use shell scripting and Python programing to access GPIO ports on Pi 3. The basic idea is to understand that we can use command lines to access and configure GPIO ports. Alternatively, as shown in the diagram, we can use shell scripting to run a Python program that can access and configure GPIO ports. Let's see how this works.

1. Access your Pi

Place your 32 Gbyte MicroSD card that you received from your instructor in the appropriate slot on the Pi. Power up the Pi using the Micro-USB adaptor. At this point you should see the RED LED. Connect the Pi to an active LAN line, using the 10/100 Ethernet LAN connector. Note that the GREEN LED on the LAN connector is should be on.

Using SSH access your Pi, as described in the previous lab. Make sure you use the login using the original username; not the one you created (*ssuee*).

Let's assume it is `192.168.1.73`. Assuming your Pi is connected to the same LAN, the IP address of the Pi is going to be within the same sub-domain (in this case 192.168.1.xx). Run the following commands:

```
ping 192.168.1.73
arp -a
```

Run a few other Linux commands to ensure everything is ok.

2. Checking your Linux & Python packages

It is important that we always check which Linux packages have been installed on our machine.

```
$ apt list # there are too many of them
$ apt list | more  # use more to see one page at a time
```

To see is SSH package has been installed do the following:

```
$ apt list -a ssh # check if SSH package has been installed
```

You can do the same thing for nmap or python packages:

```
$ apt list -a nmap; apt list – python
```

Note that in the above example we are executing two Linux commands together! The Python version you have. You can also check your python version using

```
$ python –version # you should have python version 2.x.x.or something
```

PIP is a package manager for Python. That means it's a tool that allows you to install and manage additional libraries and dependencies that are not distributed as part of the standard library. To check your pip packet manage use:

```
$ pip -version
```

For more information use `pip help`.

If pip is not installed use the following command to install it:

```
$ sudo apt install python-pip
```

3. What is PING?

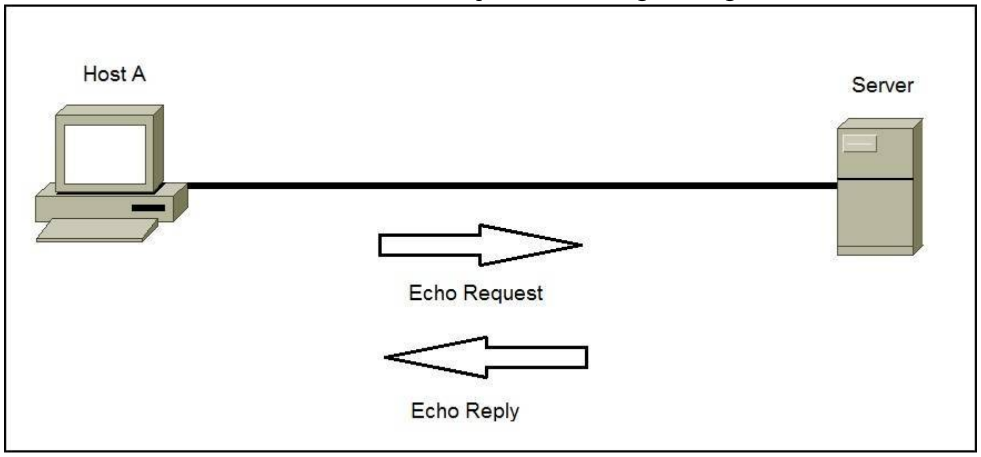
The `ping` command resolves the domain name into an IP address and starts sending ICMP packages to the destination IP. If the destination IP is reachable it will respond back and the ping command prints a line that includes the following fields:

- The number of data bytes. The default is 56, which translates into 64 ICMP data bytes - `64 bytes`.
- The IP address of the destination - `from 8.8.8.8`.
- The ICMP sequence number for each packet. `icmp_seq=1`.
- The Time to Live. – `ttl=117`
- The ping time, measured in milliseconds which is the round trip time for the packet to reach the host, and the response to return to the sender. - `time=20.9 ms`.

```
pi@raspberrypi:~ $ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=117 time=20.9 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=117 time=24.6 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=117 time=16.5 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=117 time=23.7 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=117 time=17.0 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=117 time=20.8 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=117 time=17.7 ms
^C
--- 8.8.8.8 ping statistics ---
8 packets transmitted, 7 received, 12.5% packet loss, time 23ms
rtt min/avg/max/mdev = 16.508/20.177/24.567/2.996 ms
```

By default, the interval between sending a new packet is one second. In the figure below, when we ping the server from Host A, the machine sends a packet called ECHO REQ. When the server receives the ECHO REQ, it responds with ECHO REPLY. It is possible that the server, simply ignores the ECHO REQ and does not send anything back. The `ping` command will continue to send ICMP packages to the Destination IP address until it receives an interrupt. To stop the command, just hit the `Ctrl+C` key combination.

Try pining www.sonoma.edu . What happens? Can you get the IP address of the server using ping?



The `ping` command will continue to send ICMP packages to the Destination IP address until it receives an interrupt. To stop the command, just hit the `Ctrl+C` key combination.

Here are some examples of how ping can be used (anything after # is considered to be a comment):

```
$ man ping # see the manual for ping
$ ping # see how it is used
$ ping firewall.cx
$ ping www.sonoma.edu
$ ping -c 1 8.8.8.8 # sending a single ECHO REQ
$ ping -c 1 www.google.com # sending a single ECHO REQ
$ ping -4 8.8.8.8 # ping using IPv4
$ ping -6 8.8.8.8 # ping using IPv6 - check if IPv6 is supported
$ ping -s 40 -c 5 8.8.8.8 # -s determines the size of packet in bytes
$ ping -i 2 8.8.8.8 # By default ping wait for 1 sec to send next packet we
      can change this time by using -i option.
$ ping -c 5 -p ff 8.8.8.8 # We can fill data in packet using -p option.
      Like -p ff will fill packet with ones. So, the payload (content) of
      the ICMP packets will be all FF.
```

4. What is NETSTAT?

Netstat (network statistics) is a command line tool for monitoring network connections both incoming and outgoing as well as viewing routing tables, interface statistics etc. Netstat is available on all Unix-like Operating Systems and also available on Windows OS as well. It is very useful in terms of network troubleshooting and performance measurement. netstat is one of the most basic network service debugging tools, telling you what ports are open and whether any programs are listening on ports. For more information about netstat command see footnote [1]. Make sure you practice the commands and understand what is happening.

5. What is ARP Protocol?

Address Resolution Protocol (ARP) is a procedure for mapping a dynamic Internet Protocol address (IP address) to a permanent physical machine address in a local area network (LAN). The physical machine

[1] For more information about netstat command see https://www.tecmint.com/20-netstat-commands-for-linux-network-management/

address is also known as a Media Access Control or MAC address.

The job of the ARP is essentially to translate 32-bit addresses to 48-bit addresses and vice-versa. This is necessary because in IP Version 4 (IPv4), the most common level of Internet Protocol (IP) in use today, an IP address is 32-bits long, but MAC addresses are 48-bits long [2].

Watch this video for more information: https://www.youtube.com/watch?v=cn8Zxh9bPio.

There are two types of ARP entries- **static and dynamic**. Most of the time, you will use dynamic ARP entries. What this means is that the ARP entry (the Ethernet MAC to IP address link) is kept on a device for some period of time, as long as it is being used. The opposite of a dynamic ARP entry is static ARP entry. With a static ARP entry, you are manually entering the link between the Ethernet MAC address and the IP address. Because of management headaches and the lack of significant negatives to using dynamic ARP entries, dynamic ARP entries are used most of the time.

So how is the dynamic ARP entry created? Every time you communicate with a machine with an IP address (let's say you ping a remote machine on the LAN) your machine will put that entry into its **local ARP cache** and it will stay there until the entry has not been used and the ARP cache timeout has expired. Remember, ARP only maintains LAN information.

Let's do the following exercises. First, let's reset our machine. Then, SSH into your Raspberry PI, assume my machine's IP is 10.0.0.233:

```
$ arp --version # find the version of the arp protocol
$ arp -v # show the verbos info + mac addresses
$ arp -n # show numerical addresses
$ sudo arp -d 10.0.0.1 # remove a machine from cache
$ arp -v # show the verbose info + mac addresses note 10.0.0.1 is not there
```

Let's scan my LAN:

```
$ nmap -sP 10.0.0.1/24
$ arp -v # show the verbose info + mac addresses note the added machines
```

QUESTION: Let's ping google.com. Check your arp cache. Is google.com's IP or MAC showing up? Why not?

In case you are very interested to figure out what type of devices are on your network you can use the following command to check the device type based on the OUI:

```
$ curl https://gitlab.com/wireshark/wireshark/-/raw/master/manuf | grep
D4:DC:CD
```

In the above command, cURL is simply a **curl** is used in command lines or scripts to transfer data. In This case, the data that is being transferred is located is a file called *manuf*. We are interested to grep a particular MAC address from this file[3].

[2] For more information about ARP protocol read this: https://erg.abdn.ac.uk/users/gorry/course/inet-pages/arp.html
[3] Refer to this page for the file containing OUI: https://linuxnet.ca/ieee/oui/

6. Explore Python 2.x on your Pi

In this section we explore which version of Python has been installed on your Pi and then we create a simple Python program.

- Check which version of Python is available to you using `python -V`.
- Follow these steps to create a simple Python script [4]:
  - Using `vi` command (or `nano`) create a file called `test.py` and type the following in the file:

```
#!/usr/bin/python
print "Hello, World!";
```

  - Change the mode of the file to writeable: `chmod 777 test.py`.
  - Run the script: `./test.py`.
  - What do you see?

PE0- PROGRAMING EXERCISE: Write a simple Python program to get a positive number from the use, say n and calculate the sum of the numbers from 1 to n. Your code should also determine if the final sum is an even of odd number. Your program should display something like below:

```
You entered: n
The sum of numbers form 1-n is: xxx
The sum (xxx) is an ODD/EVEN number.
```

Here is an example: let's say the user inputs 6. Then the output should be this:

```
You entered: 6
The sum of numbers form 1-6 is: 21
The sum (21) is an ODD number.
```

Practice with the test files provided

[4] For more information see https://www.raspberrypi.org/documentation/usage/python/

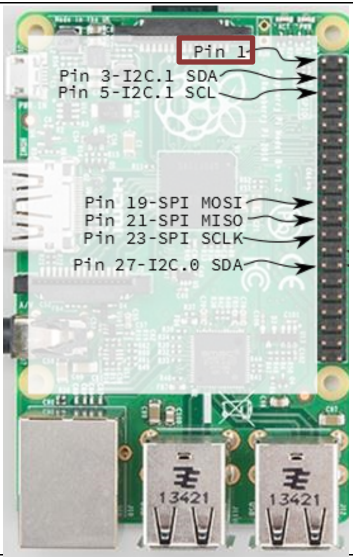
7. Using GPIO Ports

In this section we learn how to access GPIO ports on your Pi. There are 40 pins that you can access directly through the onboard connector. First enter `gpio -v` to get information about you Pi. Answer the following information:

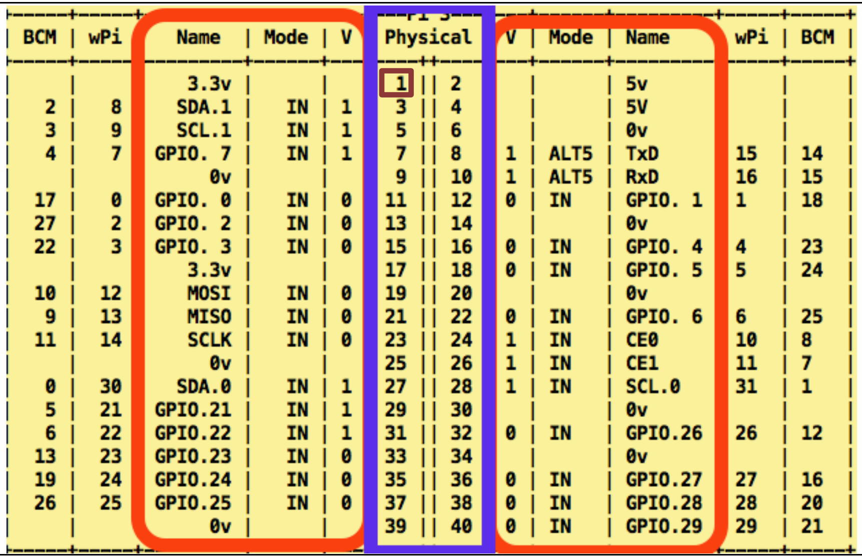
- What revision of Pi you have?
- How much memory is available to you?

Then try `gpio readall` to get information about ALL GPIO ports. You will see a table as below. See how this table is mapped on the physical pins on the board. Pay attention to pin 1. Answer the following information:

- How many GPIO ports are actually available?
- How many 5V pins are available?
- How many of the pin are ground pins?



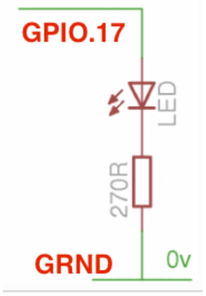
Physical Layout



```
+-----+-----+---------+------+---+--Pi 3--+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
|     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
|   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5V      |     |     |
|   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |     |     |
|   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT5 | TxD     | 15  | 14  |
|     |     |      0v |      |   |  9 || 10 | 1 | ALT5 | RxD     | 16  | 15  |
|  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
|  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
|  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
|     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
|  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
|   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
|  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
|     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
|   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
|   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
|   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
|  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
|  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
|  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
|     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
```

GPIO Pin Layout

In the above figure, note that each side has three columns. The outermost column, headed **wiringPi Pin (wPi)** refers to the pin number in the wiring Pi code. The middle one, headed **BCM** refers to the pin number of the BCM2835 chip. Also note that this is the pin number used when addressing the GPIO using the*/sys/class/gpio* interface. The innermost column, **Name** is the name of the function of the pin.



The central column contains the pin numbers on the header on the board. Pin 1 is the 3.3v power supply on the P1 connector (Rev. 1 and Rev. 2 boards), and Pin 1 is the 5v power supply on the P5 connector on Rev. 2 boards only.

If you look at the pins chart above, you can see that ***wiringPi*** pin 0 is GPIO-17. ***wiringPi*** pin 1 is GPIO-18, ***wiringPi*** pin 2 is GPIO-21 an so on. To use the GPIO pin numbering then you need to pass the -g flag to the ***gpio*** program. Let's clarify this by an example:

Connect your Raspberry Pi to an LED as suggested in the figure. Use GPIO.0=wPi.0 on your Pi.

Change GPIO.0 to an output port: `gpio mode 0 out`**.** In this case *mode 0* is referring GPIO.0. Then, run the following commands:

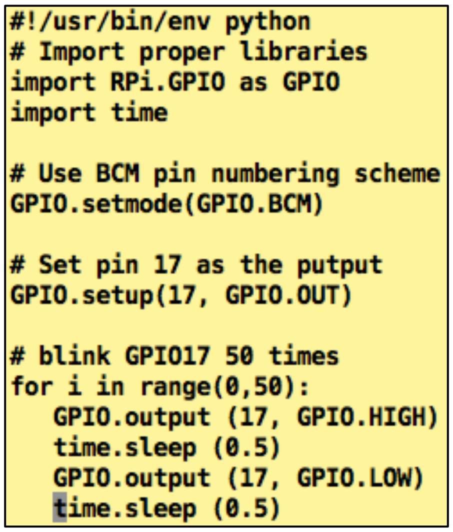
```
gpio write 0 1
gpio write 0 0
```

Note that at this point your LED should be turned ON and then turned OFF. Read the status of all the port using `gpio readall`. What is the status of GPIO.0?

It is also possible to use the g flag, referring to the BCM:

```
gpio -g write 17 1
gpio -g write 17 0
```

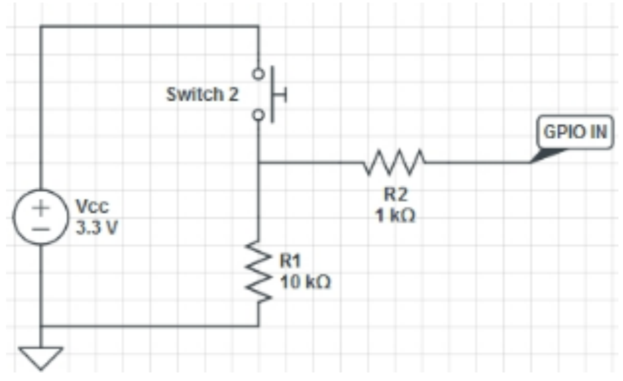
At this point create a file called *LED_blink.py* and type in the following code in there. Run the file. Your LED should be blinking.



```
#!/usr/bin/env python
# Import proper libraries
import RPi.GPIO as GPIO
import time

# Use BCM pin numbering scheme
GPIO.setmode(GPIO.BCM)

# Set pin 17 as the putput
GPIO.setup(17, GPIO.OUT)

# blink GPIO17 50 times
for i in range(0,50):
   GPIO.output (17, GPIO.HIGH)
   time.sleep (0.5)
   GPIO.output (17, GPIO.LOW)
   time.sleep (0.5)
```

Let us see how we can use a switch on the RPi. Build the circuit shown in the table below. Create a file called *switch.py* and type in the python code below. Run your program. Note that when you press the switch the terminal shows the message: `Button Pressed.` Change the program, so, when the switch is released terminal shows the message: `Button Released.`

PE1 - PROGRAMING EXERCISE: Modify the program such that when the press is switched the message shows up only once. Show the terminal display.



The Switch Circuit

```
#!/usr/bin/env python
# Import proper libraries
import RPi.GPIO as GPIO
import time

# Use BCM pin numbering scheme
GPIO.setmode(GPIO.BCM)

# Set pin 17 as the INPUT
GPIO.setup(17, GPIO.IN)
while True:
  if (GPIO.input(17)):
    print("Button Pressed")
```

Code to read the switch.

It is possible to run a Python or Shell script from a separate Python program. For example, consider the following python program that pings a server. We call this *myping.py*. We can also write simple Shell scrip, called *myping.sh.*

`#!/bin/bash`
`myhostname="google.com"`
`ping -c 1 $myhostname`
	`#!/usr/bin/python`
	`import os`
	`hostname = "google.com" #example`
	`response = os.system("ping -c 1 " + hostname)`
	
	`#and then check the response...`
	`if response == 0:`
	`  print hostname, 'is up!'`
	`else:`
	`  print hostname, 'is down!'`

myping.sh file	myping.py file executing

Any of these programs can be called from a Python program using the following commands:

```
#!/usr/bin/env python
import os
import sys

execfile('myping.py')   # run myping.py program
os.system("sh myping.sh") # run myping.sh program
```

It is also possible to write a Shell script to accept an *argument*. Run this example and see what it does (IMPORTNAT: pay attention to tabs and spaces).

```
#!/bin/bash
echo -n "Is this a good question (y/n)? "
read answer
if echo "$answer" | grep -iq "^y" ;then
    echo Yes
else
    echo No
fi
```

PE2 - PROGRAMING EXERCISE: Write a Python program that pings *google.com* only once every time an external switch is pressed. Note that you can use Linux commands when needed in your Python program.

PE3 - PROGRAMING EXERCISE: Write a Python program such that when someone pings your RPI, an LED on the Pi is turned on (this does not have to happen in real-time, but within a couple seconds the LED should be turned on). The LED should be turned off after some seconds, when pinging stops. Be prepared to demonstrate this in the lab. Note that you can use Linux commands when needed in your Python program.

PE4 - PROGRAMING EXERCISE: Write a Python program such that if the Pi has no valid IP address, the LED starts **blinking**. Be prepared to demonstrate this in the lab. Make sure you have everything you need! Note that you can use Linux commands when needed in your Python program.

Auto Lunch (optional): The following procedure allows you to run a script (say `validIpLED.py`) upon powering up the RPi[5]:

1. Make a launcher shell script (*launcher.sh* – as shown below) to run the python file.
2. Make the *launcher.sh* executable: `chmod 755 launcher.sh`
3. Add a *logs* directory to see any errors that might happen: `mkdir logs`
4. Use crontab daemon process to execute launcher script on startup: `sudo crontab -e`
5. Enter this line to bottom of crontab file:
   `@reboot sh /home/pi/Desktop/launcher.sh >/home/pi/logs/cronlog 2>&1`

8. Reboot the Pi to test the launcher script. If an error occurs it will show in the logs directory in a new file called *cronlog*

Example of *launcher.sh*

```
#!/bin/sh
# Run the validIpLED.py script on startup
cd
cd /home/ssuee/Desktop/Lab2/
sudo python validIpLED.py
cd
```

Please note that you must demonstrate PE0-PE4 in the class to your instructor to receive credit.

E. Submissions

Submit the following: Your code for PE0-PE4. In each case show a snapshot of the output. Please note that you must demonstrate PE0-PE4 in the class to your instructor to receive credit.

F. Credits

Special thanks to online resources and all SSU students who assisted putting together this lab.

G. References

[5] For more information see http://www.instructables.com/id/Raspberry-Pi-Launch-Python-script-on-startup/?ALLSTEPS