

# **RASPBERRY PI**

# **EXERCISES PACKET**

---

## PREFACE

1. Perform basic system tasks to administer your PI yourself. In order to do that, you should know about basics in the Linux OS itself including
  - o Setting your own user environment
  - o File System
  - o A bit about remote access
  - o Navigating around Linux system
  - o Administrate processes
  - o Writing Scripts
2. Work with GPIO with sensors
3. Basics in Python

To be added in the future:

Explore IOT

Ref to : online PI Tutorial Packet at <http://learn.stormingrobots.com>.

Last update: July 15th, 2017

Page 1 of 13

# CONTENTS

<b>SECTION 1 - GETTING STARTED AND MANAGE YOUR PI .....</b>	<b>3</b>
I - 1.A) Try out most commonly used commands.....	4
I - 2) Learn Vim .....	6
I - 3) Shell Exercises .....	6
<b>SECTION II - GPIO EXERCISES.....</b>	<b>8</b>
II - 1) Write three scripts :.....	8
II - 2) Write the same program in C program.....	9
<b>SECTION III - KERNEL LEVEL PROGRAMMING FOR I2C INTERFACE.....</b>	<b>11</b>
III - 1) Practice Your base conversion.....	11
III - 2) Program to communicate with your I2C devices.....	12
<b>SECTION IV – PYTHON.....</b>	<b>13</b>
IV - 1) Exercises :.....	13
IV - 2) Work with ADC to communicate with analog input device .....	13

# SECTION 1 - GETTING STARTED AND MANAGE YOUR PI

[Utilize Tutorial Section I to IV](#)

- Do's and Don'ts
- Installation
- Remote access your PI
- Know basics about the File System
- Setting your own user environment
- Navigating around the file system
- Do remote access with VNC, or ssh
- Install and deinstall software
- Know how to set up your profile / bashrc
- Know how to write basic bash script and run it
- Ping hostnames between your own computer and PI, set up hostnames, etc.
- Know basics about VIM editor and customizing it.
- Set up email sender service
- Do backup

## I - 1.A) TRY OUT MOST COMMONLY USED COMMANDS

**\*\*\*(Observe the behavior after each command. Be Inquisitive)**

```
1. man ls
2. cd /
3. pwd
4. cd ~/; pwd
5. cd .vnc; pwd
6. cd .. ;
7. ls -lS
8. ls -ltra
9. ls -ltra | more
10. ls -ltra > allFiles.txt
11. ls -l | wc -l >> allFiles.txt
12. cat "this is the end !" >> allFiles.txt
13. cat allFiles.txt
14. ls -lt all*
15. rm allFiles.txt
16. dir
17. dir -R
18. ls -lr
19. ls -lR
20. ls -lR | grep "^d"
21. alias ldir='ls -lR | grep "^d"'           #you just created a new command!
22. ldir
23. clear
24. date
25. date +%D
26. date +"%T"
27. time
28. man -? > help.txt
29. tail -f help.txt
30. ls /root/
31. sudo ls /root/
32. wget https://www.stormingrobots.com/prod/pdf/csSyllabus.pdf &           #& == runs in background
33. du
34. du -sh
35. du -h * -d 1
36. du -h * -d 3
37. du -h * | sort -n
38. du -h * | sort -nr
39. du -a . | sort -nr | head
40. du -a . | sort -nr | tail
41. df
42. df -h
43. df /home
```

```
44. echo "echo \"Hello World \" " > test.sh
45. cat test.sh
46. ls -l test.sh    # you see: -rw-r--r-- 1 ... test.sh
47. ./test.sh      # fail to run
48. chmod +x test.sh
49. ls -l test.sh   # you see: -rwxr-xr-x 1 ... test.sh
50. ./test.sh
51. ls -l | grep "^\\."    # list all hidden files and directories
52. ls -l | grep "^d"     # list directories
53. ls -l | wc -l        # count of nodes (files, directories, links, etc.)
54. ps -ef
55. watch -n 1 "ps -ef" &
56. ps                # find the PID (process ID) for this watch command
57. kill 3000         # replace 3000 with the PID of the "watch" process
58. ps                # now you see the process is gone
59. clear
60. history
61. top
62. Ctrl+Z
63. Jobs              # find the job # , e.g. [9]
64. fg 9              # type the # 9 from, the previous finding
65. bg 9              # put back to the background
    <now, you need to kill this process... >
66. find . -name lxde-pi-rc.xml
67. find . -print | grep -i lx
68. uptime
```

## —Special note about redirection ( > )

About standard file descriptors : 1, 2 == standard output and standard errors

- Syntax: command > file same as command 1> file
- echo "hello" > file == echo "hello" 1> file
- echo "hello" >&2 file == echo "hello" 1>&2 file
- application 1>/dev/null == suppress the standard output

## I - 2) LEARN VIM

- 1) Go thru the "CRASH COURSE TO LEARN VIM" in the tutorial packet.  
OR
- 2) Use : <http://www.openvim.com/tutorial.html> interactive tutorial

Go to <http://www.openvim.com/sandbox.html> to practice. When you are doing trying all the keystrokes in the crash course tutorial. Move onto the next exercise below.

## I - 3) SHELL EXERCISES

1. Tracking down where disk space has gone to . find out who has eaten up the most disk space.
2. Try out all the tutorial packet:
  - a. "COMMON ERRORS IN PARSING"
  - b. "SCRIPTS SAMPLES"
3. Create a few of aliases and put them in ~/.bash\_aliases .
4. Display "Good Morning", "Good Afternoon", or "Good Night" based on system time.
5. Read a file into bash array. Display them.
6. Create a script to produce value of factorial of "N" where N is from user input.
7. Redo (6), but make thata factorial codes as a function which takes in a single integer parameters.
8. Create a script to find all prime numbers using Sieve of Erattheosenes under N where N is from user input
9. Write a script to:
  - a. ping to find the round-trip delay to [www.google.com](http://www.google.com)
  - b. Use traceroute to see the network route taken to [www.google.com](http://www.google.com)
  - c. Run it infinitely with an interval of every 1 minute.
  - d. And start it to run in the background.
10. Kill that ping background process.
11. Schedule your PI to say hello in every 5 minutes.
12. Schedule your PI to remind you snack, and lunch time.
13. Schedule your PI to shutdown at 3:00pm, and reboot at 9:00am.
14. Write a script to monitor the growth of disk space, and schedule it to report whenever it increases by 100

Extras:

15. Customize your Prompt
16. Customize your vi environment and download the c.vim plug-in.
17. Write a script to display all the color from 16 to 255. E.g. printf "\e[48;5;24m Show This"

Raspberry Pi

```

016 017 018 019 020 021
022 023 024 025 026 027
028 029 030 031 032 033
034 035 036 037 038 039
040 041 042 043 044 045
046 047 048 049 050 051
052 053 054 055 056 057
058 059 060 061 062 063
064 065 066 067 068 069
070 071 072 073 074 075
076 077 078 079 080 081
    
```

18. Repeat Ex.9, but save the code to a file.

## SECTION II - GPIO EXERCISES

### [Utilize Tutorial Section V](#)

1. Writing scripts to manipulation basic digital devices via GPIO pins
2. Using WiringPI library to write the C programs to produce the same effect.

Special note for those who do not know C++ at all, do review the following link to at least get yourself familiar with how to read a class declaration and its methods:

Here is a simple Fruit/inventory purchase / sell class:

[https://www.element14.com/community/community/code\\_exchange/blog/2013/03/06/c-tutorial--classes](https://www.element14.com/community/community/code_exchange/blog/2013/03/06/c-tutorial--classes)

In order to stay with the scope, this is the extent of OOPS (object-oriented programming system) you need to know.

### II - 1) WRITE THREE SCRIPTS :

1. Turn on and off a LED with a push button with pull up . Display :  
 "Got Pushed... Light out" // if it is pushed  
 "Got Released... Light on" // if it is released  
 e.g. code segment

```
# gpio pin 27 to push button and a resistor to the GND
gpio -g mode 27 up
```

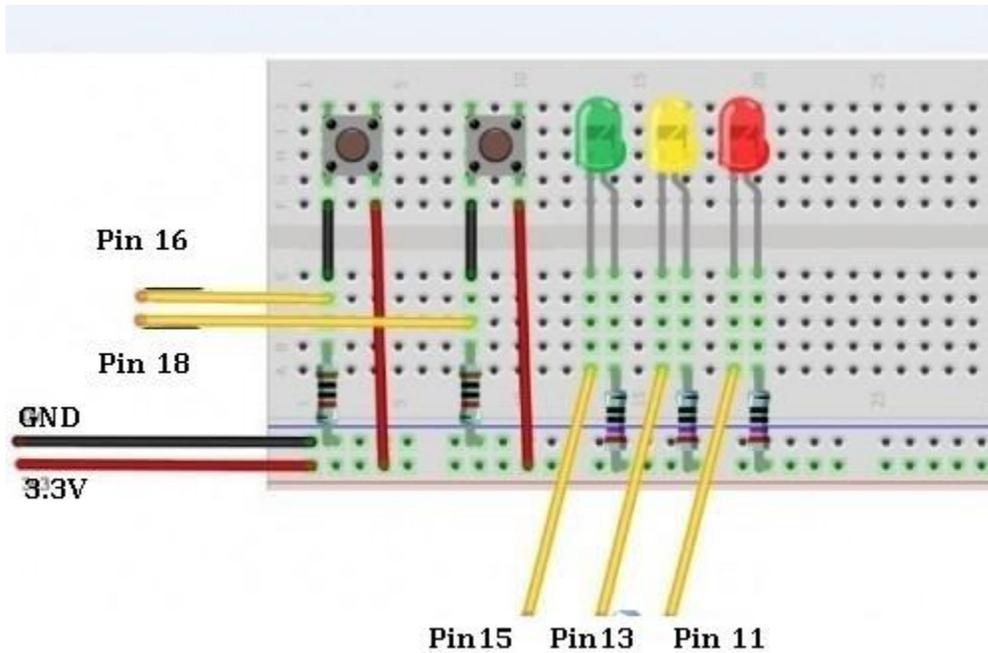
2. Turn on and off a LED with a push button with pull down. Display :  
 "Got Pushed... Light on" // if it is pushed  
 "Got Released... Light out" // if it is released

e.g. code segment

```
# gpio pin 27 to push button and a resistor to the 3V line
gpio -g mode 27 down
```

3. Take user input to determine if you want to run pull down or pull down  
e.g.

```
> runLed.sh up
> runLed.sh down
```



## II - 2) WRITE THE SAME PROGRAM IN C PROGRAM

```
#include <stdio.h> // Used for printf() statements
#include <wiringPi.h> // Include WiringPi library!

// Pin number declarations. We're using the Broadcom chip pin numbers.
const int pwmPin = 18; // PWM LED - Broadcom pin 18, P1 pin 12
const int ledPin = 23; // Regular LED - Broadcom pin 23, P1 pin 16
const int butPin = 17; // Active-low button - Broadcom pin 17, P1 pin 11

const int pwmValue = 75; // Use this to set an LED brightness

int main(void)
{
    // Setup stuff:
    wiringPiSetupGpio(); // Initialize wiringPi -- using Broadcom pin numbers

    pinMode(pwmPin, PWM_OUTPUT); // Set PWM LED as PWM output
    pinMode(ledPin, OUTPUT); // Set regular LED as output
    pinMode(butPin, INPUT); // Set button as INPUT
    pullUpDnControl(butPin, PUD_UP); // Enable pull-up resistor on button
```

```
printf("Blinker is running! Press CTRL+C to quit.\n");

// Loop (while(1)):
while(1)
{
    if (digitalRead(butPin)) // Button is released if this returns 1
    {
        pwmWrite(pwmPin, pwmValue); // PWM LED at bright setting
        digitalWrite(ledPin, LOW); // Regular LED off
    }
    else // If digitalRead returns 0, button is pressed
    {
        pwmWrite(pwmPin, 1024 - pwmValue); // PWM LED at dim setting
        // Do some blinking on the ledPin:
        digitalWrite(ledPin, HIGH); // Turn LED ON
        delay(75); // Wait 75ms
        digitalWrite(ledPin, LOW); // Turn LED OFF
        delay(75); // Wait 75ms again
    }
}

return 0;
}

gcc -Wall blinker.c -I wiringPi blinker
```

sudo ./blinker

## SECTION III - KERNEL LEVEL PROGRAMMING FOR I2C INTERFACE

[Utilize Tutorial Section VI](#)

— Writing Device System Calls to access I2C devices

### III - 1) PRACTICE YOUR BASE CONVERSION

Decimal (base-10) to Hexadecimal (base-16) and Binary (base-2)

Decimal (base-10)	Hexadecimal (base-16)	Binary (base-2)
12		
14		
13		
15		
16		
127		
255		
256		
1023		
1024		
2047		
2048		
65535		
65536		

Binary (base-2) to Hexadecimal (base-16) and Decimal (base-10)

Binary (base-2)	Hexadecimal (base-16)	Decimal (base-10)
00001		
00011		
00111		
01001		
110011		
1001110		
00001001		
10001001		
01001001		

10101000		
----------	--	--

### III - 2) PROGRAM TO COMMUNICATE WITH YOUR I2C DEVICES

Hook up the following i2c sensors and display all the following:

What to test with	What you need to do :
HiTechnic IR Seeker	<ul style="list-style-type: none"> <li>• Get version</li> <li>• # from all zones. Tested with an IR Ball</li> </ul>
Mindsensors light array	<ul style="list-style-type: none"> <li>• Get version</li> <li>• # from all 8 light sensors. Tested with a lines</li> </ul>
Mindsensors I2C adapter connected to ev3 light	<ul style="list-style-type: none"> <li>• Get the raw data</li> </ul>
Mindsensors I2C adapter	<ul style="list-style-type: none"> <li>• Get version</li> <li>• Modify the I2C address of the module</li> </ul>

## SECTION IV – PYTHON

[Utilize Tutorial Section VII](#)

Know the basic data types and structure about Python 3.X

### IV - 1) EXERCISES :

- Array : must use "List"
- Recursion
- More on List and Tuples

From <http://learn.stormingrobots.com>

### IV - 2) WORK WITH ADC TO COMMUNICATE WITH ANALOG INPUT DEVICE

Work with With adafruit ADC

- 1) Now you will write a python program to detect black & white using the ADC and photocell .
  - Write the photocell and create a voltage divider
  - Program to detect white or black by using hardcoded threshold
- 2) Do (2) , except using adjustable threshold by using the potentiometer.
  - Photocell
  - Potentiometer
  - Program the potentiometer to create adjustable threshold