# Internship Project Installation Guide

Giuseppe Cervone
giuseppe.cervone.971@gmail.com

## 1 Introduction

The following installation guide will break down how to set up the specific devices, which software has to be installed and the configuration parameters to follow for a complete installation of the remote network monitoring system.

## 2 Devices

### 2.1 Raspberry Pi

As far as choosing the best Raspberry Pi model for the job, the parameters to look for are that it needs to have enough RAM to handle possible big data chunks, it needs to have a fast enough CPU, it can possibly can be used without cooling, everything at a low enough cost. Needing for it not to be cooled removes the Raspberry Pi 4 from the equation, since it prefers having a cooling solution and the increased clock speeds won't help with serial communication limitations, on the other hand the Raspberry Pi 2 already comes with 1Gb Ram, but misses the clock speed needed for multiple programs to run at a good enough speed. The Raspberry Pi 3 models should be able to handle the data size and the processes we want to run. The following is the configuration used for the testing of this project:

- Raspberry Pi 3 Model B Ver1.2:
  - Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
  - 1GB RAM
  - BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
  - 100 Base Ethernet
  - 40-pin extended pinout with GPIO support
  - 4 USB 2 ports
  - 4 Pole stereo output and composite video port
  - Full size HDMI
  - CSI camera port for connecting a Raspberry Pi camera
  - DSI display port for connecting a Raspberry Pi touchscreen display
  - Micro SD port for loading your operating system and storing data
  - Upgraded switched Micro USB power source up to 2.5A
  - Raspberry Pi OS Lite (latest version)
  - 16GB microSD card

Using the RPI-Imager tool, install the latest Raspberry PI OS Lite image. If you don't want to use a monitor or keyboard to run the first configuration, it is possible to enable WiFi and SSH capabilites at this stage. If you want to do so, after having created the image, include two files in the *boot* folder on the Raspberry Pi SD card. The first file, is a *wpa_supplicant.conf* file, used to enable WiFi. An example of this type of file can be found here below:

```
country=<Insert 2 letter ISO 3166-1 country code here>
update_config=1

network={
 ssid="<Name of your wireless LAN>"
 psk="<Password for your wireless LAN>"
}
```

For SSH support, it's enough to include a file named *ssh* in the boot folder, without extensions and without anything written inside. Upon first boot up, the default Raspberry Pi login credentials are username: pi, password: raspberry.
As Raspberry Pi OS Lite is an OS without a desktop environment, getting to system settings is done using the utility *raspi-config*. Using the command *sudo raspi-config*, you open the utility, once in the main menu change these specific settings:

- In the system options:
  - Enable WiFi if needed for SSH purposes and if you haven't done this before. On first boot country code is also set.
  - Change username and password if needed.
- In the interface options:
  - Enable SSH if you haven't done this before. Using SSH is suggested for development on these units, as they are installs without a desktop environment/window manager configuration, and thus it's best to use your preferred terminal emulator.
  - Use the setting *Serial interface*, remembering to disable login shell but enable serial interface.
- In the localisation options:
  - Change the the timezone to the local one.
- Disable bluetooth:
  - It's suggested disabling bluetooth to make the final product more secure.
  - Disable bluetooth by adding the line *dtoverlay=disable-bt* in */boot/config.txt* using your preferred command line editor.
- Additional software:
  - Run *sudo apt-get update && sudo apt-get upgrade*
  - Run *sudo apt-get install python3* to install Python in it's latest version.
  - Run *sudo apt-get install python3-pip* to install Python's package manager.
  - Run *pip3 install pyserial* to install the serial library used to have the machines comunicate.
  - Run *pip3 install pyzabbix* to install the Python library used to interface with the Zabbix API. This packet only has to be installed on the Raspberry working on the private network.

– Run *sudo apt-get install zabbix-sender* to install Zabbix sender, the utility we use to interface with Zabbix trapper items. This packet only has to be installed on the Raspberry Pi machine localized on the public network.
– Run *sudo apt-get install git* to install the git program. Useful to clone into the directory of this software.

### 2.2 Serial cable

The cable used for this project is a TTL-232R-3V3 cable, which is USB-to-serial, with +3.3V TTL levels UART signals. The cable has 6-pins on one end, and USB on the other. With the cable plugged in as shown in Figure 1, run the command *ls /dev/tty** on both Raspberry Pi machines to check that the ports used to send and receive data are available. The port that's sending data will be */dev/ttyS0*, while for the other port it should be */dev/ttyUSB0*. In older Pi models, instead of using */dev/ttyS0*, the port that was used instead to access the pins was the port */dev/ttyAMA0*. This change in naming caused incompatibility in some programs, thus Raspian will always map ports *S0* and *AMA0* to the alias */dev/serial0*, so it is suggested to use the alias in programming. As mentioned earlier in Figure 1 there is an image showing how the cable is meant to be plugged in, while in the Appendix 1 there is the full extract from the documentation of the cable, explaining the functionality of all the cable pins in detail.

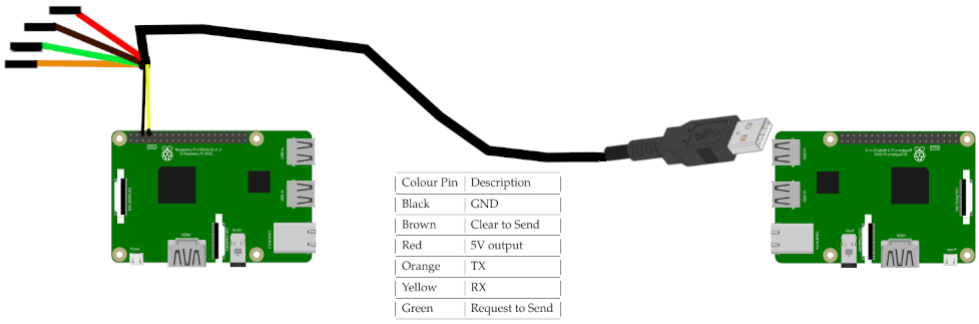


Figure 1: Pinout diagram

The cable pinout is peculiar as most pins are not used. The pins that get used are the ground pin and the pin that sends data over to the other Raspberry Pi machine. Yellow pin is plugged in GPIO14, ground can be plugged in any of the Raspberry Pi ground pins.

## 3 Serial interface

### 3.1 Diagnostic test

With the current setup ready, use the diagnostic test found in the folder *DiagnosticTest* to check if the two Raspberry Pi are connected correctly. The code can be found both in Appendix 2 and in the folder, there is the example code and example output that can be used to troubleshoot and check that the cable has been plugged in correctly. Make sure to run the code *sndtest.py* on the machine where the GPIO pins are being used, and *rcvtest.py* has to be run on the Raspberry Pi where the USB is plugged in. It's important that you start by running the command *python3 rcvtest.py*, and then run *python3 sndtest.py* on the other Raspberry Pi. In case this process has issues, refer to the list of possible fixes reported here.

- Check that UART is enabled in */boot/config.txt* by adding *enable_UART=1*.

- Check that serial console is disabled by removing lines *console=serial0,115200* or *console=ttyS0,115200* in */boot/cmdline.txt*.
- Check for permissions in the dialout group.
- Check that */dev/serial0* doesn't have a getty console running on it. In case it does, it can be disabled by using the commands: *sudo systemctl stop serial-getty@ttyS0.service* and *sudo systemctl disable serial-getty@ttyS0.service*.

# 4 Zabbix

## 4.1 Installation guide

Zabbix is the network monitoring tool of choice. For the initial version of the project, version 5.4 was used, running on a Apache web server and MySQL database. It is suggested to follow the installation guide found on the Zabbix website specifically for our Raspberry Pi OS version. The installation guide can be found both at the link *zabbix.com/download* or following these commands below:

```
# wget https://repo.zabbix.com/zabbix/5.4/raspbian
/pool/main/z/zabbix-release/zabbix-release_5.4-1+debian10_all.deb

# sudo dpkg -i zabbix-release_5.4-1+debian10_all.deb

# sudo apt update

# sudo apt install zabbix-server-mysql zabbix-frontend-php
zabbix-apache-conf zabbix-sql-scripts zabbix-agent
```

Create now the initial database using these commands:

```
# sudo apt-get install mariadb-server
# sudo mysql_secure_installation
```

During the secure installation set a new root password and reload privilege tables. then proceed with these commands:

```
# mysql -uroot -p
# password
# mysql> create database zabbix character set utf8 collate utf8_bin;
# mysql> create user zabbix@localhost identified by 'password';
# mysql> grant all privileges on zabbix.* to zabbix@localhost;
# mysql> quit;
```

Using the preferred password for the Zabbix user. Then proceed with this command to initialize the Zabbix database tables:

```
# zcat /usr/share/doc/zabbix-sql-scripts/mysql/create.sql.gz |
mysql -uzabbix -p zabbix
```

And then configure the database for Zabbix server by adding the password to */etc/zabbix/zabbix_server.conf* with the parameter *DBPassword=password*.

Before having access to the GUI, using the following commands

```
# systemctl restart zabbix-server zabbix-agent apache2
# systemctl enable zabbix-server zabbix-agent apache2
```

will reload the new settings. Having followed the guide on both machines, access to the frontends should be available by connecting to the URL *http://RaspberryPiIPadress/zabbix*. You will be greeted by the welcome screen, where you can pick the installation language. The next screen is a pre-requisites screen, where Zabbix should find no problems. Now you configure the DB connection, here it is important to insert the password we used earlier to authenticate in MySQL. The other parameters are supposed to look like the ones in the picture below.

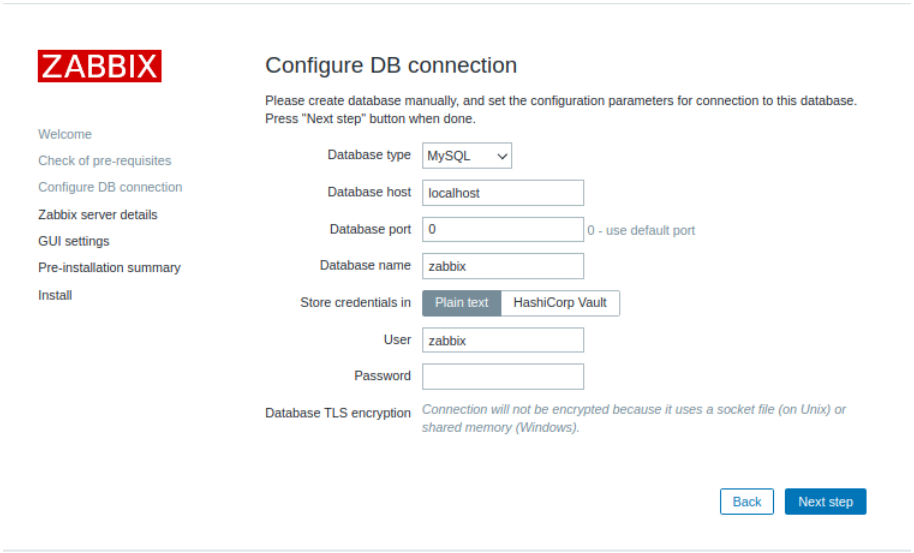


Figure 2: Zabbix server parameters

The next screen shows a series of server settings, here you can give a name to the Zabbix installation but it is not mandatory. In the last part of the installation pick the correct timezone. The Zabbix frontend will now be correctly set-up.

## 4.2 Configuration adjustments

Remembering that the default login credentials are *user: Admin* and *password: zabbix*, login to the GUI, and apply the following adjustments via the frontend:

- Raspberry Pi private network adjustments:
  - Change the name of the host *Zabbix server* to *Zabbix server pvt*.
- Raspberry Pi external network adjustments:
  - Add new host *Zabbix server pvt* without filling the interface parameter.

## 4.3 Hosts and items

It is possible to add more hosts to the configuration, each host is equivalent to a machine you want to monitor. For a host to be monitored, a Zabbix agent instance has to be installed and configured on it. On *zabbix.com/download_agents* you can find agent install files for most platforms. Once the hosts are configured in the private Zabbix instance, it is then possible to add items, equivalents to the parameters the client wants to track. For a seamless integration, every host and item that is added in the private network, that we want to also see on the public network, has to be configured on the Zabbix instance running in the public network, without the interface parameter and the exact same name. In the same way, every item configured on the private network, has to be configured on the public network, but with the Zabbix trapper type.

### 4.4 Triggers and actions

Zabbix handles atypical behaviour via a system of triggers and actions. The idea behind this system is the following: The user writes an expression regarding one or more parameters being monitored, if the expression is true at any moment, the trigger is activated and the user is notified in the dashboard. An action can be configured as a reaction to one or more triggers being activated at the same moment, and an action can include a variety of possibilites such as automated commands or notifications via email. The possibilites regarding triggers and the actions connected to the activation of those triggers is endless, as Zabbix allows for a ton of customization options. Use the guide written below to get a general idea of a basic configuration including the configuration of a trigger, linking that trigger to an action, and the configuration of a mail server, so as to receive a notification of the trigger being activated. For the configuration parameters, the Zabbix documentation has a high amount of information in great detail, thus it is highly suggested to follow that for further information.

1. Go in the Configuration menu and select hosts.
2. In the hosts section, click on *Triggers* on the host for which we want to configure a trigger for, then in the window where all the triggers for the items are shown, click on the button *Create trigger*.
3. In this section there are a series of mandatory parameters to fill:
   - Name: The name of the trigger, mandatory parameter.
   - Severity: Severity of the trigger. For our use case we will use WARNING.
   - Expression: An explanation of the value to monitor, and the parameter for which Zabbix has to set a trigger. In our case the expression will be: $last(Host/Item) > value$.
   - OK event generation: The action that must happen for the trigger to go back to the OK state. Here it is possible to set more expressions to further specify the trigger values. In our case we will leave it as *Expression*
   - The other parameters we won't be using in this case, however the parameters are very well explained in the Zabbix documentation. Leave them as default in this case.
4. Having set up the trigger, set up a corresponding action by moving, via the *Configuration* menu, to *Actions*, option *Trigger actions*.
5. In this page, all of our triggers can be viewed with their status. A trigger creates an event, and the action is done based on an event.
6. To create a new action, click on *Create action*. This will open a configuration window with two tabs, *Action* and *Operations*.
7. In the *Action* tab, fill in the *Name* parameter with what the specific action is connected to. In the *Conditions* section, select the type of condition with which the action should be activated. This section is also really customizable, but for the sake of the demonstration use *Trigger* as type, with operator *Equals*, choosing the trigger we built earlier.
8. In the operations tab there are a lot of customizable options too. For this guide simply add a basic operation, which sends to a specified user an email. This step gives us the liberty of resolving automatically the problem without sending an email.

9. The last part of this guide involves filling the parameters for the mail server. To do so open the *Administration* tab, and find the option *Media types*.
10. Find the Email parameter and click on it, then based on your Email client fill in the specific SMTP parameters. These are the parameters from which the email will be sent from.
11. Now in User settings section, go in the profile subsection. On the top there will be a *Media* tab, click on it, and add the media where you will be receiving the Email from the Zabbix server.

Having followed this guide, a basic trigger with a basic action should be configured. Zabbix allows for many more customization options, and even for the possibility to automate the problem solving aspect of the warnings. All the information to do so can be found on the official Zabbix documentation, which is explained much more in depth.

# 5 File transfer setup

## 5.1 Python scripts

Having imported the repo in the home directory of the Raspberry Pi on both private and public networks, inside the repo there should be a */home/pi/DocInternship/send/main.py* file and a */home/pi/DocInternship/rcv/main.py* file. The content of both of these scripts can be found in Appendix 3. The code which runs on the Raspberry Pi sending data will work under a cron timer of five minutes, and will send data over the last five minutes, using as a starting time parameter, the time found in the file *time.txt*. The receiving script will open at startup, listening for data on the serial port, receiving all the data every time it's being sent, divide it in files of 250 lines for *zabbix_sender*, and then the script will push the data to the trapper items.

## 5.2 Cron jobs

Automation of the send and receive scripts is done via cron jobs, here below is the configuration parameters for both Raspberry Pi machines.

### 5.2.1 Private Raspbery Pi

Use command crontab -e to open up the cron job editor, if asked to choose an editor pick the editor you prefer. Include at the end of the file the following command **/5 * * * * /usr/bin/python3 /home/pi/DocInternship/send/main.py*. Save and close, making sure that the command has been written correctly using *crontab -l*. This command will have the sender job every 5 minutes. The other command used by cron is the following *@reboot sudo rm /home/pi/DocInternship/send/time.txt*. This command has the job of removing the file *time.txt* on boot. The reasoning behind the use of this command is the following, with the device being turned off, no data is being captured by Zabbix. The implication of this data missing, is that file transfers starting from that moment will include no data for a long time, thus it makes more sense to delete the file, and have our code create a new file with the current timestamp.

## 5.3 Public Raspberry Pi

Use command crontab -e to open up the cronjob editor, if asked to choose an editor pick the editor you prefer. Include at the end of the file the following command *@reboot /usr/bin/python3 /home/pi/DocInternship/rcv/main.py*. Save and close, making sure that the command has been written correctly using *crontab -l*.

# 6 Useful features

At this point, the system should be working in it's entirety. It is to be considered that Zabbix as a program, and the system as a whole, gives the user the chance for a lot more customization options and usability changes. Below we have a couple of examples.

## 6.1 Monitoring

Both the send and the receive processes will be running as orphan processes. It is possible to monitor their cron timings by using the commmand *tail -f /var/log/syslog*. The python scripts both also print debug statements on a log file, which can be found by using the command *tail -f /home/pi/DocInternship/send or receive/logger.log*.

## 6.2 Zabbix

To make the public Zabbix interface easier to use there are a couple of customization options you can consider. For example Zabbix gives you the opportunity to configure the dashboard to include the most important parameters. The possibility of installing an Android app to access push notification and dashboard is also available. Worthy of note is also the possibility of receving push notifications to telegram for alarms, as opposed to receving them via email.

# 7 Appendix

## 7.1 Appendix 1: Documentation extract

### 4.2 TTL-232R-5V and TTL-232R-3V3 Cable Signal Descriptions

| Header Pin Number | Name | Type | Colour | Description |
|---|---|---|---|---|
| 1 | GND | GND | Black | Device ground supply pin. |
| 2 | CTS# | Input | Brown | Clear to Send Control input / Handshake signal. |
| 3 | VCC | Output | Red | +5V output, |
| 4 | TXD | Output | Orange | Transmit Asynchronous Data output. |
| 5 | RXD | Input | Yellow | Receive Asynchronous Data input. |
| 6 | RTS# | Output | Green | Request To Send Control Output / Handshake signal. |

**Table 4.1 TTL-232R-5V and TTL-232R-3V3 Cable Signal Descriptions**

## 7.2 Appendix 2: Diagnostic output and code example

Listing 1: sndtest.py

```
#!/usr/bin/env python
import time
import serial

ser = serial.Serial(
        port='/dev/serial0',
        baudrate = 9600,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
        timeout=1
)
counter=0

while True:
        ser.write(b'Write counter: %d \n'%(counter))
        time.sleep(1)
        counter += 1
```

Listing 2: rcvtest.py

```
#!/usr/bin/env python
import time
import serial

ser = serial.Serial(
        port='/dev/ttyUSB0',
        baudrate = 9600,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
        timeout=1
)

while True:
        x=ser.readline()
        print(x)
```

Listing 3: output.txt

```
b'Write counter: 0\n'
b'Write counter: 1\n'
b'Write counter: 2\n'
b'Write counter: 3\n'
b'Write counter: 4\n'
b'Write counter: 5\n'
```

### 7.3 Appendix 3: Data transfer

Listing 4: main.py

```
from pyzabbix import ZabbixAPI
import sys
import datetime
import time
import logging
import argparse
import os
import serial
import hashlib

def calculateHash():
    f = open("/home/pi/DocInternship/send/data.txt", "rb")
    sha256_hash = hashlib.sha256()
    line = f.read(1024)
    while line:
        sha256_hash.update(line)
        line = f.read(1024)
    sha256_hash = sha256_hash.digest()

    return sha256_hash

def sendFile(ser):
    eof = b'EOF'
    f = open("/home/pi/DocInternship/send/data.txt","rb")
    logging.info("Starting transfer...")

    line = f.read(512)
    while line:
        ser.write(line)
        line = f.read(512)
    f.close()
    ser.write(eof)

    logging.info('Transfer completed...')

def createSerial():
    try:
        ser = serial.Serial(
            port = "/dev/serial0",
            baudrate = 115200,
            parity = serial.PARITY_NONE,
            stopbits = serial.STOPBITS_ONE,
            bytesize = serial.EIGHTBITS,
            timeout=1
        )
        return ser
    except Exception as e:
```

```
        logging.warning(e)


def setDate():
    if os.path.isfile('/home/pi/DocInternship/send/time.txt'):
        f = open('/home/pi/DocInternship/send/time.txt', 'r')
        timeFrom = int(f.readline())
        timeTill = timeFrom + 300
        f.close()
        f = open('/home/pi/DocInternship/send/time.txt', 'w')
        f.write(str(timeTill))
    else:
        timeTill = int(time.time())
        timeFrom = timeTill - 300
        f = open('/home/pi/DocInternship/send/time.txt', 'w')
        f.write(str(timeTill))
        f.close()
    return timeFrom, timeTill

def historyToFile(zapi, hosts, items):
    y = 0
    timeFrom, timeTill = setDate()

    f = open('/home/pi/DocInternship/send/data.txt', 'w')

    for item in items:
        hostname = hosts[y]["name"]
        hostid = hosts[y]["hostid"]
        for x in range(len(item)):
            itemkey = item[x]['key_']
            itemid = item[x]['itemid']
            itemtype= item[x]['value_type']
            historys = zapi.history.get(hostids = hostid,
                    itemids = itemid, time_from = timeFrom,
                    time_till = timeTill, history = itemtype,
                    output="extend")
            for history in historys:
                f.write('"%s" %s %s %s\n' % (hostname, itemkey,
                            history["clock"], history["value"]))
        y+=1
    logging.info('Exported history...')
    f.close()


def getItems(zapi, hosts):
    items = []
    for host in hosts:
        hostid = host['hostid']
        items.append(zapi.item.get(hostids = hostid,
            output=["key_", "hostid", "hostname","value_type"]))
```

```
    if len(items) == 0:
        logging.critical('No items... Quitting program')
        sys.exit()
    else:
        logging.info('Items found...')
        return items


def getHosts(zapi):
    hosts = zapi.host.get(output=['name'])
    if len(hosts) == 0:
        logging.critical('No hosts... Quitting program')
        sys.exit()
    else:
        logging.info('Hosts found...')
        return hosts


def login(zapi, username, password):
    try:
        zapi.login(username, password)
        logging.info("Login Success...")
    except:
        logging.critical("Zabbix server not reachable... Quitting program")
        sys.exit()

def main():
    logging.basicConfig(filename='/home/pi/DocInternship/send/logger.log',
                        format='%(asctime)s %(message)s', level=logging.INFO)

    zapi = ZabbixAPI('http://192.168.1.198/zabbix')

    login(zapi, 'Admin', 'zabbix')

    hosts = getHosts(zapi)

    items = getItems(zapi, hosts)

    historyToFile(zapi, hosts, items)

    ser = createSerial()

    sendFile(ser)

    hash1 = calculateHash()
    time.sleep(5)
    ser.write(hash1)
    logging.info('Hash sent... Closing program.')
```

```
if __name__ == '__main__':
    main()
```

Listing 5: main.py

```
import serial
import time
import hashlib
import subprocess
import logging

def sender():
    f = open("/home/pi/DocInternship/rcv/data.txt", "r")
    line = f.readline()
    while line:
        x = 0
        f2 = open("/home/pi/DocInternship/rcv/tmp.txt", "w")
        while x in range(250):
            f2.write(line)
            line = f.readline()
            x+=1
        f2.close()
        subprocess.run(["zabbix_sender", "-z", "192.168.1.157", "-i",
                "/home/pi/DocInternship/rcv/tmp.txt", "-T", "-vv"])


def calculateHash():
    f = open("/home/pi/DocInternship/rcv/data.txt", "rb")
    sha256_hash = hashlib.sha256()
    line = f.read(1024)
    while line:
        sha256_hash.update(line)
        line = f.read(1024)
    sha256_hash = sha256_hash.digest()

    return sha256_hash


def recvHash(ser):
    while True:
        if ser.inWaiting() < 32:
            time.sleep(1)
        else:
            break
    hash2 = ser.read(32)
    return hash2


def recvFile(ser):
    eof = b'EOF'
    f = open("/home/pi/DocInternship/rcv/data.txt", "wb")
```

```
    while True:
        recvdatalen = ser.inWaiting()
        if recvdatalen>0:
            line = ser.read(recvdatalen)
            if eof in line:
                f.write(line[:-3])
                break
            else:
                f.write(line)
        time.sleep(0.001)
    logging.info("File transfer completed...")


def createSerial():
    try:
        ser = serial.Serial(
            port = "/dev/ttyUSB0",
            baudrate = 115200,
            parity = serial.PARITY_NONE,
            stopbits = serial.STOPBITS_ONE,
            bytesize = serial.EIGHTBITS,
            timeout = 1
        )
        logging.info("Serial port created...")
        return ser
    except Exception as e:
        logging.warning(e)


def main():

    logging.basicConfig(filename='/home/pi/DocInternship/rcv/logger.log',
                        format='%(asctime)s %(message)s', level=logging.INFO)

    ser = createSerial()

    while True:
        recvFile(ser)

        hash2 = recvHash(ser)

        hash1 = calculateHash()

        if hash1==hash2:
            logging.info("File Transfer Success... Importing data in Zabbix...")
            sender()
            logging.info("File imported...")
        else:
            logging.warning("File Transfer Failed")
```

```
if __name__ == '__main__':
    main()
```