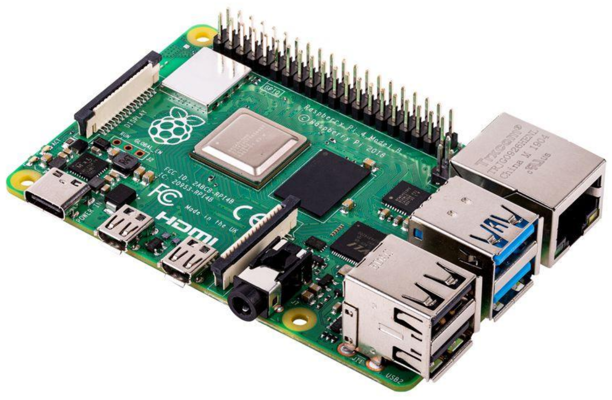
IOTEDU



RASPBERRY PI WITH THE INTERNET OF THINGS - IOTEDU

Chapter 2 – Introduction to Linux Commands, MQTT and Python

Introduction to Linux Commands

The Linux command line is a text interface to your computer. Often referred to as the shell, terminal, console, prompt or various other names, it can give the appearance of being complex and confusing to use. Every Linux system includes a command line of one sort or another. This tutorial includes some specfic steps for Ubuntu 18.04 but most of the content should work regardless of your Linux distribution.

About Linux:

During the formative years of the computer industry, one of the early operating systems was called Unix. It was designed to run as a multi-user system on mainframe computers, with users connecting to it remotely via individual terminals. These terminals were pretty basic by modern standards: just a keyboard and screen, with no power to run programs locally. Instead they would just send keystrokes to the server and display any data they received on the screen.

Each of these tasks required its own program or command:

- one to change directories (cd)
- another to list their contents (ls)
- a third to rename or move files (mv)

The original Unix shell program was just called sh, but it has been extended and superceded over the years, so on a modern Linux system you're most likely to be using a shell called **bash**.

Linux is a sort-of-descendent of Unix. The core part of Linux is designed to behave similarly to a Unix system, such that most of the old shells and other text-based programs run on it quite happily. In theory you could even hook up one of those old 1970s terminals to a modern Linux box, and access the shell through that. But these days it's far more common to use a software terminal: that same old Unix-style text interface, but running in a window alongside your graphical programs.

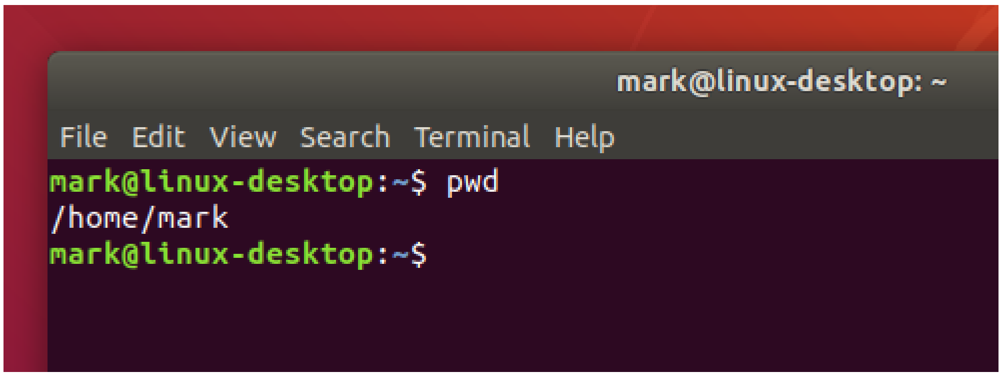
Opening a terminal

On a Ubuntu 18.04 system you can find a launcher for the terminal by clicking on the Activities item at the top left of the screen, then typing the first few letters of "terminal", "command", "prompt" or "shell". Yes, the developers have set up the launcher with all the most common synonyms, so you should have no problems finding it.

Let's run our first command. Click the mouse into the window to make sure that's where your keystrokes will go, then type the following command, all in lower case, before pressing the Enter or Return key to run it.

```
pwd
```

You should see a directory path printed out (probably something like /home/YOUR_USERNAME), then another copy of that odd bit of text.



```
mark@linux-desktop:~$ pwd
/home/mark
mark@linux-desktop:~$
```

Now to the command itself. pwd is an abbreviation of 'print working directory'. All it does is print out the shell's current working directory. But what's a working directory?

You can change the working directory using the `cd` command, an abbreviation for 'change directory'. Try typing the following:

cd /

pwd

The "/" directory, often referred to as the root directory, is the base of that unified file system.

From the root directory, the following command will move you into the "home" directory (which is an immediate subdirectory of "/"):

cd home

pwd

To go up to the parent directory, in this case back to "/", use the special syntax of two dots (..) when changing directory (note the space between cd and .., unlike in DOS you can't just type cd.. as one command):

cd ..

pwd

Typing cd on its own is a quick shortcut to get **back to your home directory:**

cd

pwd

You can also use .. more than once if you have to move up through multiple levels of parent directories:

cd ../..

pwd

Notice that in the previous example we described a route to take through the directories. The path we used means "starting from the working directory, move to the parent / from that new location move to the parent again". So if we wanted to go straight from our home directory to the "etc" directory (which is directly inside the root of the file system), we could use this approach:

```
cd

pwd

cd ../../etc

pwd
```

Making Folders

```
mkdir /tmp/tutorial

cd /tmp/tutorial
```

Notice the use of an absolute path, to make sure that we create the tutorial directory inside /tmp. Without the forward slash at the start the mkdir command would try to find a tmp directory inside the current working directory, then try to create a tutorial directory inside that. If it couldn't find a tmp directory the command would fail.

In case you hadn't guessed, **mkdir is short for 'make directory'**. Now that we're safely inside our test area (double check with pwd if you're not certain), let's create a few **subdirectories**:

```
mkdir dir1 dir2 dir3
```

There's something a little different about that command. So far we've only seen commands that work on their own (cd, pwd) or that have a single item afterwards (cd /, cd ~/Desktop). But this time we've added three things after the mkdir command. Those things are referred to as parameters or arguments, and different commands can accept different numbers of arguments. The mkdir command expects at least one argument, whereas the cd command can work with zero or one, but no more. See what happens when you try to pass the wrong number of parameters to a command:

Let's take a look at them with the ls (list) command:

ls

If you've followed the last few commands, your terminal should be looking something like this:

The terminal, after running mkdir and ls

Notice that mkdir created all the folders in one directory. It didn't create dir3 inside dir2 inside dir1, or any other nested structure. But sometimes it's handy to be able to do exactly that, and mkdir does have a way:

mkdir -p dir4/dir5/dir6

ls

This time you'll see that only dir4 has been added to the list, because dir5 is inside it, and dir6 is inside that. Later we'll install a useful tool to visualise the structure, but you've already got enough knowledge to confirm it:

cd dir4

ls

cd dir5

ls

cd ../..

The "-p" that we used is called an option or a switch (in this case it means "create the parent directories, too").

Enter the following commands to try out different ways to create folders with spaces in the name:

mkdir "folder 1"

mkdir 'folder 2'

mkdir folder\ 3

mkdir "folder 4" "folder 5"

mkdir -p "folder 6"/"folder 7"

ls

Although the command line can be used to work with files and folders with spaces in their names, the need to escape them with quote marks or backslashes makes things a little more difficult. You can often tell a person who uses the command line a lot just from their file names: they'll tend to stick to letters and numbers, and use underscores ("_") or hyphens ("-") instead of spaces.

Creating files using redirection

Suppose we wanted to capture the output of that command as a text file that we can look at or manipulate further. All we need to do is to add the greater-than character (">") to the end of our command line, followed by the name of the file to write to:

ls > output.txt

This time there's nothing printed to the screen, because the output is being redirected to our file instead. If you just run ls on its own you should see that the output.txt file has been created. We can use the cat command to look at its content:

cat output.txt

Okay, so it's not exactly what was displayed on the screen previously, but it contains all the same data, and it's in a more useful format for further processing. Let's look at another command, echo:

echo "This is a test"

Yes, echo just prints its arguments back out again (hence the name). But combine it with a redirect, and you've got a way to easily create small test files:

```
echo "This is a test" > test_1.txt
echo "This is a second test" > test_2.txt
echo "This is a third test" > test_3.txt
ls
```

You should cat each of these files to theck their contents. But cat is more than just a file viewer - its name comes from 'concatenate', meaning "to link together". If you pass more than one filename to cat it will output each of them, one after the other, as a single block of text:

```
cat test_1.txt test_2.txt test_3.txt
```

Moving Files

That's good, but perhaps the choice of backup name could be better. Why not rename it so that it will always appear next to the original file in a sorted list. The traditional Unix command line handles a rename as though you're moving the file from one name to another, so our old friend mv is the command to use. In this case you just specify two arguments: the file you want to rename, and the new name you wish to use.

```
mv backup_combined.txt combined_backup.txt
ls
```

Another way

```
mv "folder 1" folder_1
mv "folder 2" folder_2
mv "folder 3" folder_3
```

mv "folder 4" folder_4

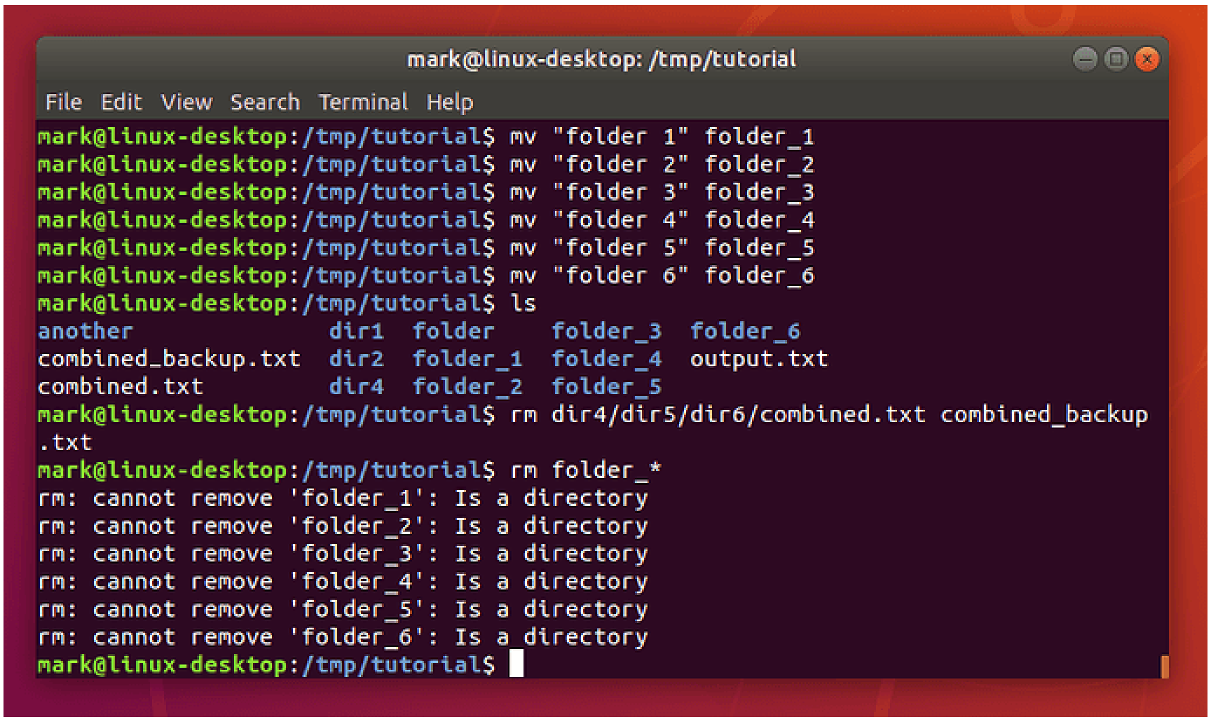
mv "folder 5" folder_5

mv "folder 6" folder_6

ls

Perhaps we should remove some of those excess directories as well:

rm folder_*


mark@linux-desktop: /tmp/tutorial
File Edit View Search Terminal Help

```
mark@linux-desktop:/tmp/tutorial$ mv "folder 1" folder_1
mark@linux-desktop:/tmp/tutorial$ mv "folder 2" folder_2
mark@linux-desktop:/tmp/tutorial$ mv "folder 3" folder_3
mark@linux-desktop:/tmp/tutorial$ mv "folder 4" folder_4
mark@linux-desktop:/tmp/tutorial$ mv "folder 5" folder_5
mark@linux-desktop:/tmp/tutorial$ mv "folder 6" folder_6
mark@linux-desktop:/tmp/tutorial$ ls
another              dir1  folder    folder_3  folder_6
combined_backup.txt  dir2  folder_1  folder_4  output.txt
combined.txt         dir4  folder_2  folder_5
mark@linux-desktop:/tmp/tutorial$ rm dir4/dir5/dir6/combined.txt combined_backup
.txt
mark@linux-desktop:/tmp/tutorial$ rm folder_*
rm: cannot remove 'folder_1': Is a directory
rm: cannot remove 'folder_2': Is a directory
rm: cannot remove 'folder_3': Is a directory
rm: cannot remove 'folder_4': Is a directory
rm: cannot remove 'folder_5': Is a directory
rm: cannot remove 'folder_6': Is a directory
mark@linux-desktop:/tmp/tutorial$ 
```

What happened there? Well, it turns out that rm does have one little safety net. Sure, you can use it to delete every single file in a directory with a single command, accidentally wiping out thousands of files in an instant, with no means to recover them. But it won't let you delete a directory. I suppose that does help prevent you accidentally deleting thousands more files, but it does seem a little petty for such a destructive command to balk at removing an empty directory. Luckily there's an rmdir (remove directory) command that will do the job for us instead:

rmdir folder_*

Linux commands in Raspberry Pi

When you open the Raspberry Pi terminal, the first thing to start with is to grab the upgrade of the OS. Then the update begins. Or, somehow you will need to install some packages. For these purposes, you will use Linux commands more often.

After reading this article, you may feel that Linux is not that tough and you may get well versed in the usage. I have updated the list restricted to the use of Raspberry Pi, but remember that Linux has a huge base of commands.

General Commands

apt-get

Installs, upgrades and uninstalls packages.

First, we will learn to install a package or a language maybe.

1. For example, Pylint is a package for the Python and for installing the package you have to give the above command.

sudo apt-get install pylint

2. This will list the packages it wants to install, tell you how much space it needs for the download, and then get on with it when you tell it to. Sometimes, it asks for Yes/No for upgrading the system. You have to press either Y or N to continue the upgrade.

sudo apt-get upgrade

3. To make sure the system is updated.

sudo apt-get update

4. To clear the previous commands and start over from the beginning

clear

5. To Shutdown the system

poweroff

6. To open the configuration in the Raspberry Pi

raspi-config

7. To reboot the Raspberry Pi

sudo reboot

How do I use sudo last command?

Simply enter the command again adding sudo before the command.

Press Up arrow to get the last command and put sudo in front of it.

Enter sudo !!

8. To shutdown the Pi immediately

shutdown -h now

9. To shutdown the Pi at a specific time period at 11:20 AM

shutdown –h 11:20

10. Opens the Linux text Editor with the new file name file.txt

nano file.txt

11. To find out who you are logged is as

whoami

12. To take a screenshot, that saves in the pi folder.

scrot

13. Lists the currently logged in users

w

14. Obtain information on the Linux computer with the kernel details.

uname -a

uname -r

uname -v

uname -s

-a an option to see everything

-s to see the kernal name

-r to view the release of the kernel

-v to view the kernel version

15. To set a password for the other user and self. for other user, just use sudo in front of the command.

sudo passwd iot

Here “iot” is the other user

File System

1. To list the files and folders in the current directory

ls

To list the files and folders in the current directory with a detailed listing (long)

ls -l

To include hidden files use the -a (all files) option

ls -lha

2. echo

The echo command prints (echoes) a string of text to the terminal window.

```
echo iot4beginners
```

The echo command can show the value of environment variables, for example, the $USER, $HOME, and $PATH environment variables.

```
echo $PATH
echo $USER
echo $HOME
echo iot4beginners >> out.txt
```

“iot4beginners” is written and saved as a file named out.txt

3. Creates a new directory named iot4beginners inside the current directory.

```
mkdir iot4beginners
```

4. To display the contents of the file

```
cat out.txt
```

5. To change the current directory of the some other directory

```
cd /home/pi
```

6. To remove or delete the file

```
rm out.txt
```

7. To create a new file in the current directory

```
touch man.txt
```

8. Removes the entire directory

```
rmdir iot4beginners
```

9. To copy the file or the entire directory

```
cp /home/pi/out.txt /home/pi/Magenta/
```

Networking Commands

1. To check the status or the ip address of the wired and wireless connection

```
ifconfig
```

2. To check the status of the wireless adapter

```
iwconfig
```

3. To create a static IP address for the Ethernet adapter

```
sudo ip a add 192.168.88.xx/24 dev eth0
```

2. MQTT Message Queuing Telemetry Transport)

What is an MQTT?

The MQTT (MQ Telemetry Transport or previously known as the Message Queuing Telemetry Transport) is a light weight publish/subscribe protocol designed for M2M (Machine to Machine) telemetry in low bandwidth environments.

Initially MQTT was designed by IBM and Arcon in 1999 for Oil Pipeline Telemetry Systems over the satellites. Nowadays MQTT is one of the main messaging protocols of the Internet of Things.

MQTT is suitable for the transport of telemetry data ie., data from the sensors and actuators.

MQTT vs HTTP

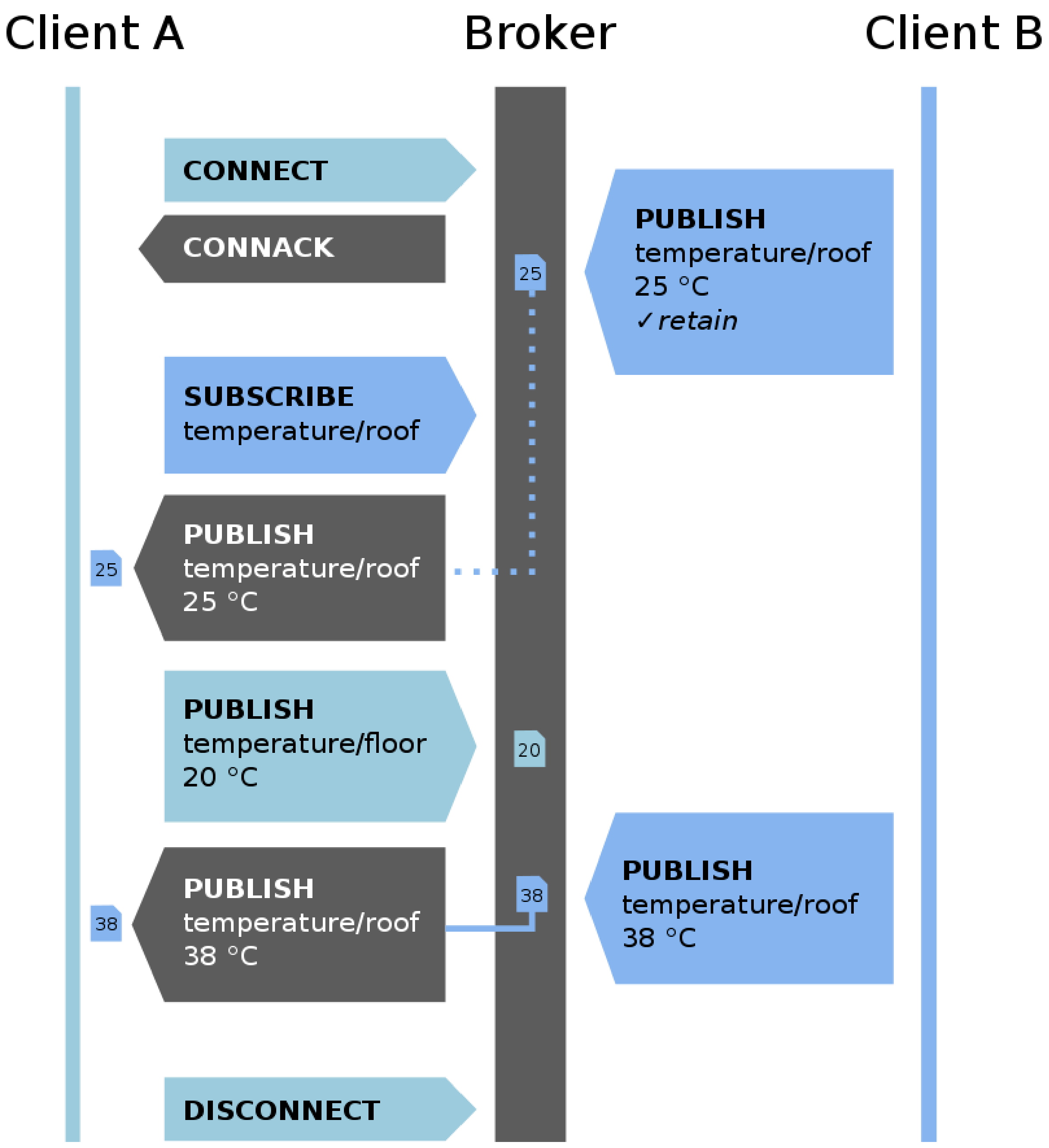
HTTP is the most popular used messaging protocol but in recent years, HTTP has been slowly replaced by MQTT by the IoT developers. The reason is that MQTT is data-centric whereas HTTP is document centric. HTTP is a request-response protocol for client-server computing and does not go along with mobile devices.

Besides, in comparison to HTTP, MQTT Protocol ensures high delivery guarantees. There are 3 levels of Quality of Services:

– at most once: guarantees a best effort delivery.

– at least once: guaranteed that a message will be delivered at least once. But the message can also be delivered more than once.

– exactly once: guarantees that each message is received only once by the counterpart


Client A
Broker
Client B
CONNECT
CONNACK
PUBLISH temperature/roof 25 °C ✓retain
25
SUBSCRIBE temperature/roof
PUBLISH temperature/roof 25 °C
25
PUBLISH temperature/floor 20 °C
20
PUBLISH temperature/roof 38 °C
38
38
PUBLISH temperature/roof 38 °C
DISCONNECT

MQTT Client

You need to assign addresses to the client you like to work with the messaging systems. For MQTTv3.1.1 there is client software available in almost all programming languages and for the main operating systems Linux, Windows, Mac from the Eclipse Paho project. Here is a link to the client comparison chart and download page.

MQTT Publish/Subscribe

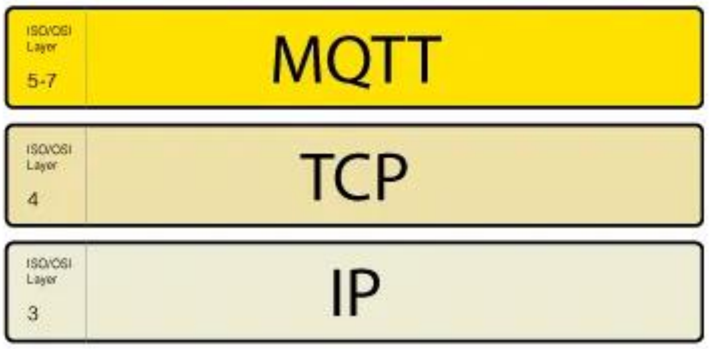
In a publish/subscribe scheme, the client publisher publishes the data to the client subscriber which has subscribed to the publisher under a topic. For example, a client subscriber receives the data (sensor data) from the client publisher which contains the sensor data.

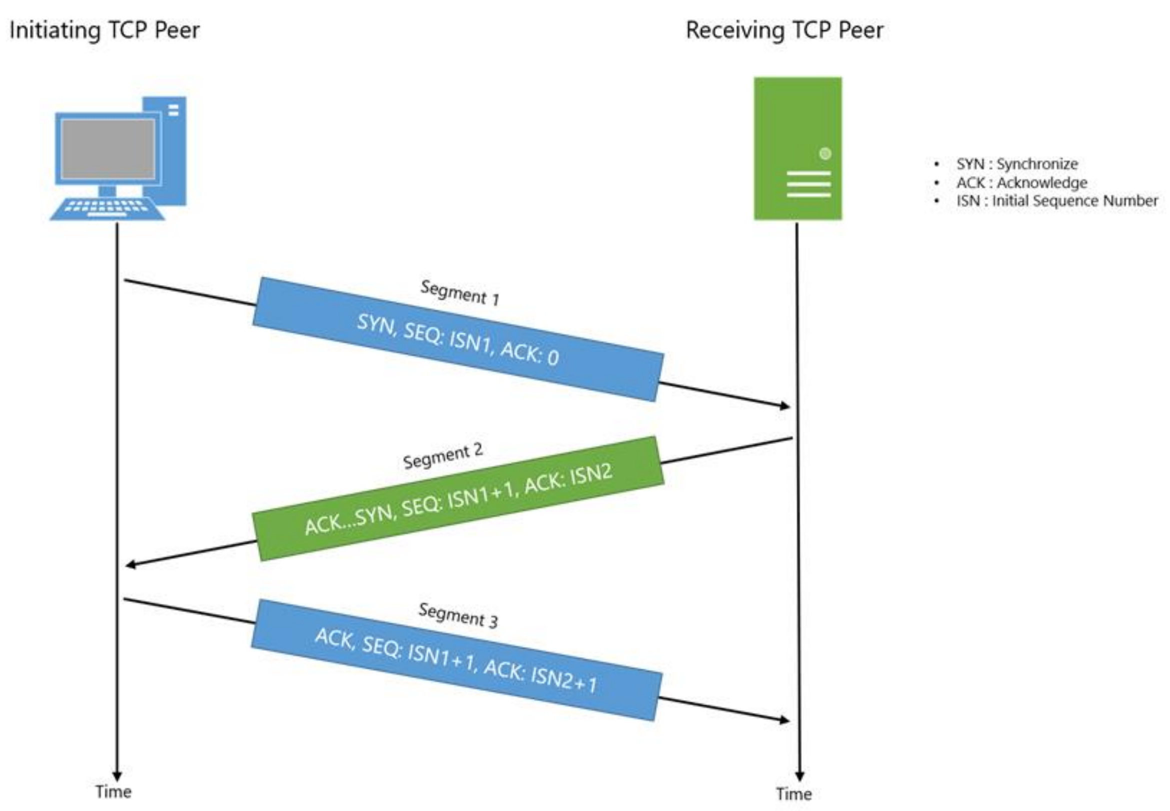
MQTT Broker

The MQTT broker is the heart of the publish/subscribe protocol. It acts as the hub for the protocol as it handles up to thousands of connected MQTT clients. The responsibility of the broker is to authenticate and authorize the client. It receives all the messages and filters them and sends the message to the corresponding subscriber. The broker also consists of session details and missed messages.

MQTT Connection

The MQTT client always connects and communicates with the broker. It is based on TCP/IP and both the client and the broker need to have this stack.





Installing Mosquitto MQTT on Raspberry Pi

First update the operating system on your Raspberry Pi:

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

Now, open a terminal and type the following command:

```
sudo apt install -y mosquitto mosquitto-clients
```

You can test your installation using the following command:

```
mosquitto -v
```

This will return the version of Mosquitto MQTT installed on your Raspberry Pi.

For the rest of this tutorial, you shall need the IP address of your Raspberry Pi. Type the following command and note down the IP address:

```
hostname -I
```

Creating an MQTT Broker on Raspberry Pi

The easiest way to understand this protocol is to create a broker on Raspberry Pi and use it to publish and subscribe to topics. Mosquitto MQTT provides a layer of security that authorizes only specific clients to publish or subscribe to topics. For this, we need to set up a username and password. This step is optional, however, it is recommended to use it in all your projects.

Type the following command:

```
sudo nano /etc/mosquitto/mosquitto.conf
```

The last line in the file will be:

```
include_dir /etc/   osquito/conf.d
```

Remove this line and add the following lines at the end of the file:

```
allow_anonymous false
password_file /etc/mosquitto/pwfile
listener 1883
```

The above three lines will tell the broker, listening on port 1883, to prevent any communications from devices that do not have a valid username and password.

The above step is optional. Hence, to proceed without adding a username and password, remove the first two lines and directly go to the rebooting part.

```
sudo mosquitto_passwd -c /etc/mosquitto/pwfile username
```

Type the above command in a terminal window. Replace "username" with your username. You will be prompted to enter a password. Type a password and press Enter.

Finally, reboot the Pi for the changes to take effect.

```
sudo reboot
```

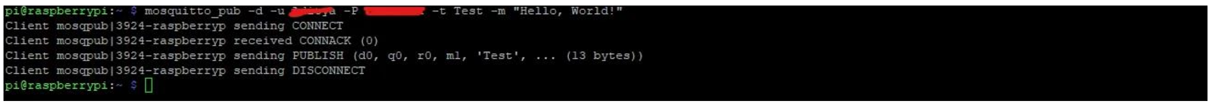
Publish a message to a Topic

To publish a message to a topic, type the following command:

mosquitto_pub -d -u username -P password -t Test -m "Hello, World!"

Replace username and password with you username and password. In case that step was skipped, simply type the following command:

mosquitto_pub -d -t Test -m "Hello, World!"



```
pi@raspberrypi:~ $ mosquitto_pub -d -u [redacted] -P [redacted] -t Test -m "Hello, World!"
Client mosqpub|3924-raspberryp sending CONNECT
Client mosqpub|3924-raspberryp received CONNACK (0)
Client mosqpub|3924-raspberryp sending PUBLISH (d0, q0, r0, m1, 'Test', ... (13 bytes))
Client mosqpub|3924-raspberryp sending DISCONNECT
pi@raspberrypi:~ $
```

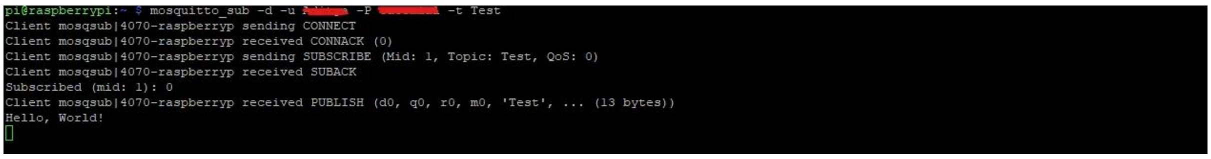
Subscribe to a Topic

Open a terminal and type the following command:

mosquitto_sub -d -u username -P password -t Test

In the command, replace username and password with the username and password you created before. In case that step was skipped, just type the following command:

mosquitto_sub -d -t Test
We have successfully subscribed to our Test topic. Now we have to publish a message to this topic.



```
pi@raspberrypi:~ $ mosquitto_sub -d -u [redacted] -P [redacted] -t Test
Client mosqsub|4070-raspberryp sending CONNECT
Client mosqsub|4070-raspberryp received CONNACK (0)
Client mosqsub|4070-raspberryp sending SUBSCRIBE (Mid: 1, Topic: Test, QoS: 0)
Client mosqsub|4070-raspberryp received SUBACK
Subscribed (mid: 1): 0
Client mosqsub|4070-raspberryp received PUBLISH (d0, q0, r0, m0, 'Test', ... (13 bytes))
Hello, World!
```