adafruit learning system

Cloud Cam: Internet-Connected Security Camera

Created by Tony DiCola

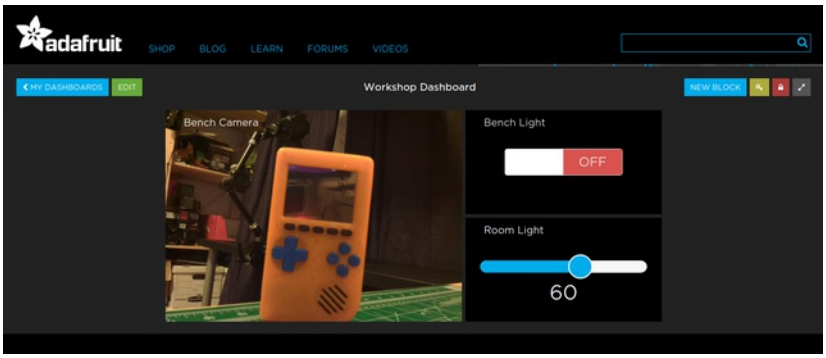
Last updated on 2017-07-05 07:35:54 PM UTC

Guide Contents

Overview

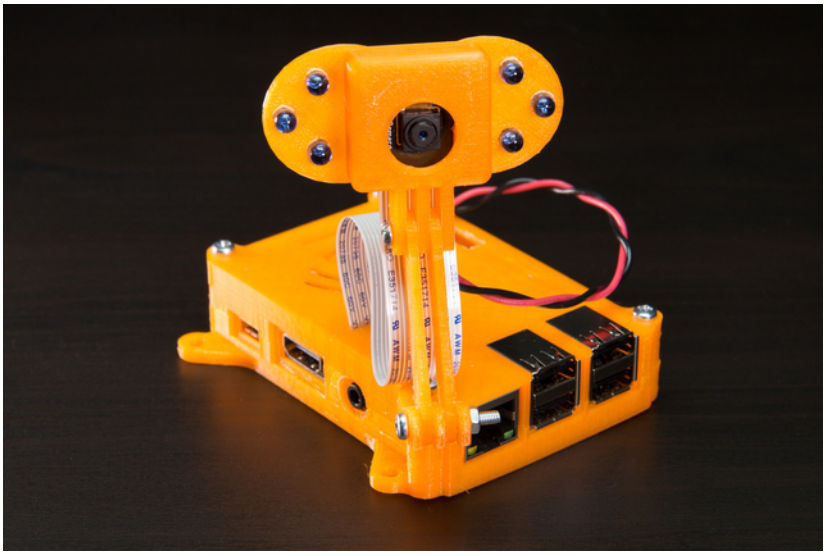
The cloud cam is a project to build an internet-connected security camera using a Raspberry Pi and Pi camera. The camera can detect motion and upload images to Dropbox's cloud storage service, or the beta of Adafruit IO (http://adafru.it/jcY), Adafruit's internet of things service.

For example build a fancy dashboard to watch and control the lights in a room:



Or keep an eye on your pets and track when they eat, even in the dark:

What will you watch with the cloud cam project?



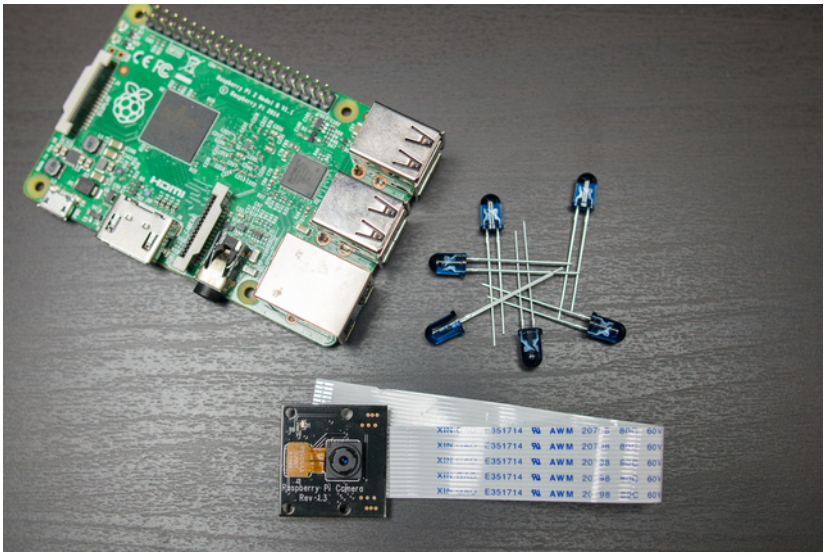
To build this project you'll want to be familiar with the basics of using the Raspberry Pi. Check out the following guides if you're new to the hardware:

- Adafruit Raspberry Pi Lessons (http://adafru.it/jcW) - Check out lesson 1 through 3 to get started and connect your Raspberry Pi to your network and the internet. Also read lesson 6 to learn how to connect to the Pi's command terminal with SSH.
- Raspberry Pi Camera Documentation (http://adafru.it/jcX)

You might also need to do a little bit of soldering to wire IR LEDs for illumination in the dark. If you're new to soldering check out Adafruit's guide to excellent soldering (http://adafru.it/dxy). This is an easy soldering project for any skill level.

Continue on to learn about the parts and hardware used in this project.

Hardware



Parts

You'll need the following parts to build this project:

- **Raspberry Pi** - Any model will do, including the Raspberry Pi 2 (http://adafru.it/2358), B+ (http://adafru.it/1914), A+ (http://adafru.it/2266), and even older original models like the A and B (http://adafru.it/998).
- **Raspberry Pi Camera** - I recommend the Pi NoIR camera (http://adafru.it/1567) which can detect infrared (IR) light and see in darkness (with illumination from IR LEDs), but the normal Pi camera (http://adafru.it/1367) will work great too.
- **Optional: Longer Pi camera cable (http://adafru.it/2087)** - The cable that comes with the Pi camera is fine for positioning it near the Pi, however if you need to move the camera further from the Pi look into a longer cable.
- If using the NoIR camera to see in darkness you'll need a source of infrared light like from IR LEDs. To build the 3D printed Pi camera holder with IR LEDs shown in this project you'll need:
  - **6x super bright IR LEDs (http://adafru.it/388)**
  - **2x 24 ohm 1/4 watt resistors** - These resistors limit current through the LEDs to around 100 milli-amps total. You can use higher ohm value resistors at the expense of dimmer LEDs, however you **must** have some resistors.
  - **Optional: Wide angle camera lens adapter (http://adafru.it/jcZ)** - You might want a wide angle lens if the field of view of the Pi camera is too narrow for your needs.
  - **2x wires with a female pin adapter** (http://adafru.it/1952) - These will connect the IR LEDs to the Pi GPIO pins for powering the LEDs.
  - **Solder and soldering tools (http://adafru.it/136)**
- **Miniature WiFi Module (http://adafru.it/814)** - Not required if your Pi can connect to your network with an ethernet cable.
- **Power supply (http://adafru.it/1995)** - Use a good quality 1.5-2 amp 5V power supply to power the Pi.
- **Raspberry Pi case (http://adafru.it/2285)** - Although not required a case will help protect your Pi. Make sure the case has a slot to allow the camera cable and GPIO pins to be accessed! If you have a 3D printer you can even 3D print a nice Pi case.
- **MicroSD card (http://adafru.it/1294)** - You'll need an 8GB card to run the latest Raspbian 'Jessie' operating system used in this project.

Raspberry Pi Setup

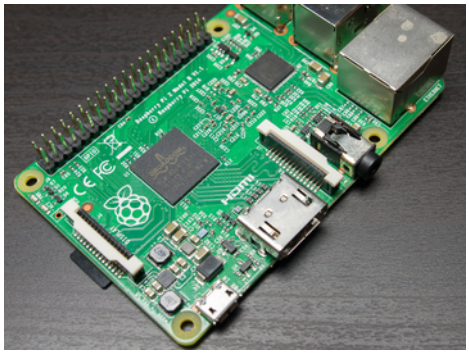
Start by burning the latest **Raspbian Jessie** operating system to a SD card for the Pi. You can find the latest image from the Raspberry Pi foundation here (http://adafru.it/fQi) (remember you want the **Jessie** version!). To burn the image to a SD card follow the official instructions here (http://adafru.it/jd0) or this great guide on the topic (http://adafru.it/jd1).

This guide is written to use the latest Jessie version of the Raspbian operating system. Be sure to use this newer version or else the later software installation steps won't work!

If you're using a WiFi adapter with the Pi follow this guide on how to setup the adapter (http://adafru.it/jd2) and connect it to your WiFi network.
Before continuing to the next steps make sure your Pi has internet access either through a wired or a wireless connection.

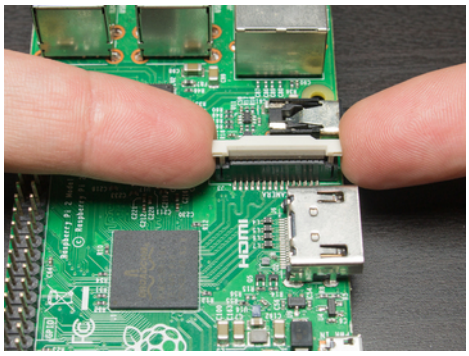
Pi Camera Setup

To setup the Pi camera carefully follow the steps below:

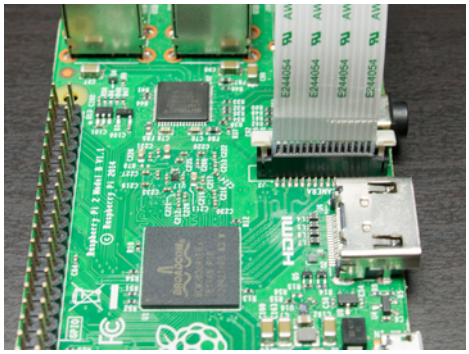
- 

  First make sure the Pi is turned off, then locate the Pi camera connector on the board. The Pi camera connector is the long white connector between the HDMI port and ethernet adapter. The connector is marked with the word CAMERA on the board.

  Be careful not to use a similar connector at the end of the board above the SD card holder. This connector is for the official Pi LCD display and will not work with the camera!

- 

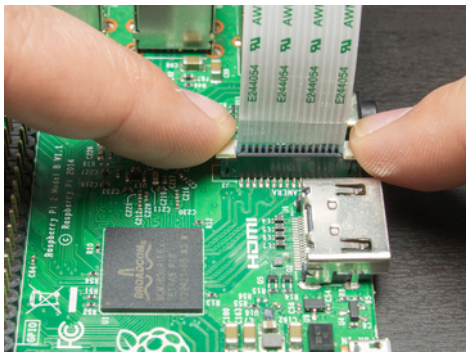
  Gently pull up on the ends of the white plastic connector to open it. The connector can swing back after it raises up.

- 

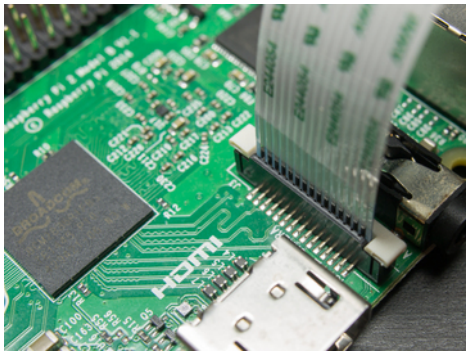
  Insert the camera cable with the metal pads on the cable facing towards the metal pads inside the connector. The blue tape on the back of the cable should be facing towards the ethernet port above it.

  Make sure the cable inserts all the way into the connector until you feel it touch the bottom of the board. The cable should be straight and level, not crooked or at an angle. Also ensure the connection is free of dust, debris, hair, etc.

  Make sure to plug the cable in as shown or else the camera will not work!

- 

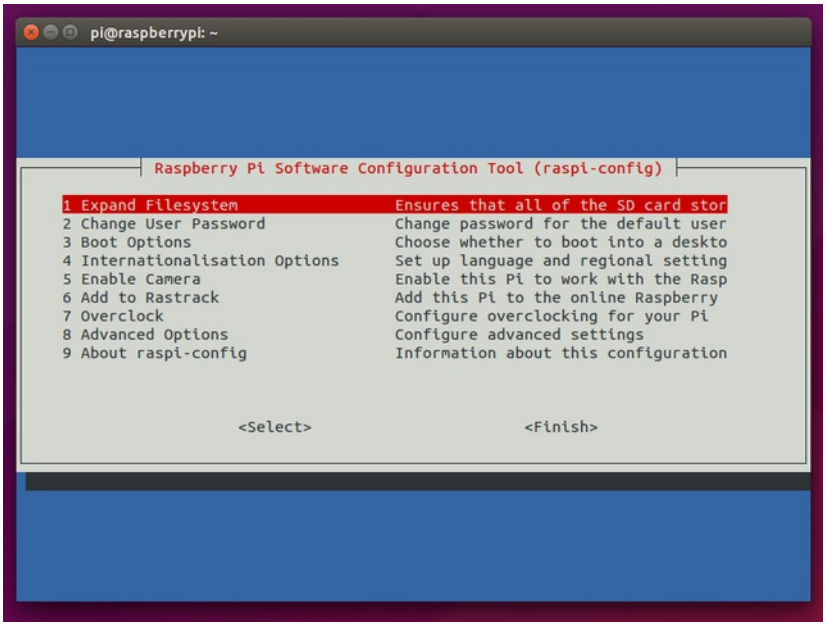
  Gently press down on the ends of the white connector until it slides back into the closed position. The cable should be firmly attached to the Raspberry Pi.

-  The cable should look like this once connected.

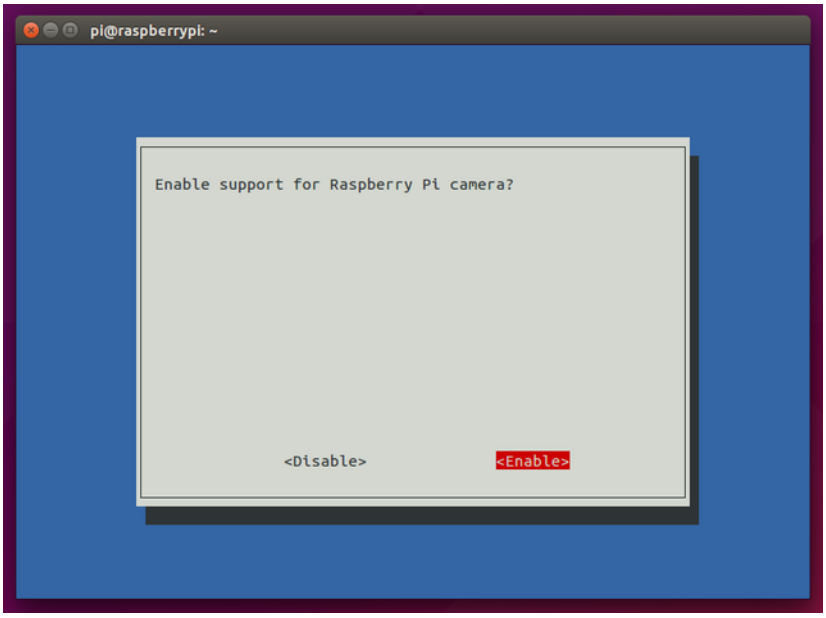
Once the Pi camera is connected power on the Pi and connect to it in a command line terminal. You will now need to enable the camera by using the raspi-config tool. After logging in to the Pi run the following command:

```
sudo raspi-config
```

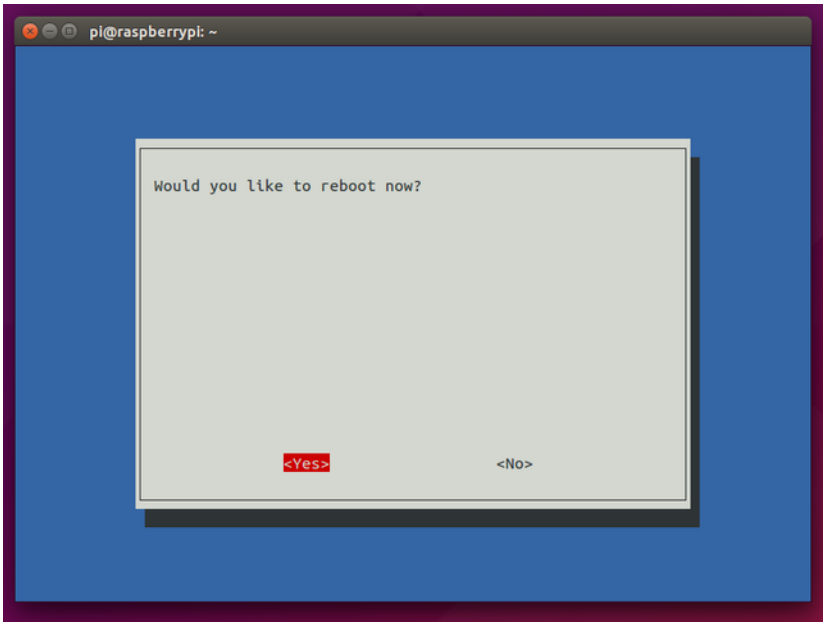
Once the raspi-config tool loads you will see a screen similar to the following:



Choose the **Enable Camera** option, then confirm the selection by choosing**Enable** in the dialog that appears:



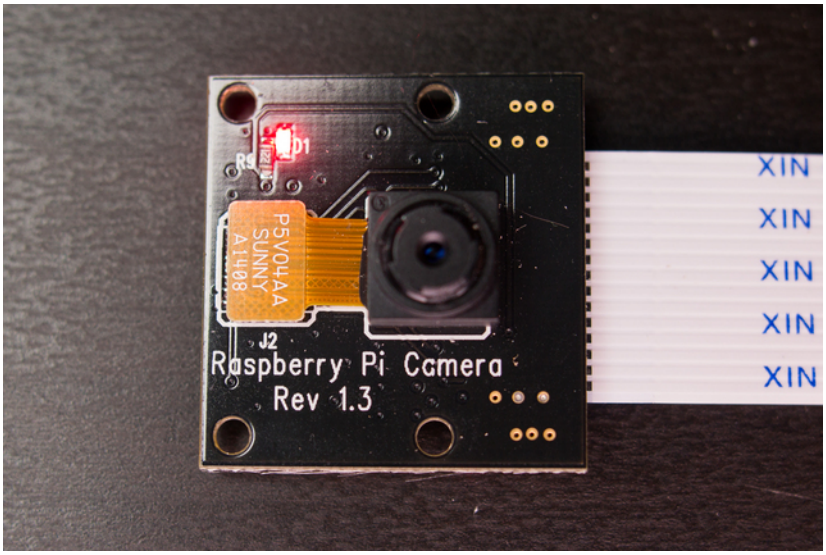
Now exit the tool by selecting the **Finish** option from the main menu. In the dialog that appears asking to reboot select the **Yes** option so the Pi reboots.



After the Pi reboots log back in and test that the camera works by using the raspistill command (http://adafru.it/jd3). Run the following command:

```
raspistill -o cam_test.jpg
```

You should see the red LED on the camera board light up as a photo is taken:



If you see the command fail with an error go back and carefully check you have enabled the camera using raspi-config, and that the camera cable is firmly connected to both the Pi and camera board. Sometimes the cable and connector on the camera board itself needs to be removed and reinserted to get a good connection. Also try running the rpi-update utility (http://adafru.it/jd4) to update the Pi firmware and try again.

After the raspistill command successfully runs you should be able to copy the cam_test.jpg off the Pi and view it on your computer. Congrats the Pi camera is setup and ready to use!

NoIR Filter Camera

The Pi NoIR filter camera (http://adafru.it/1567) is a special version of the Pi camera that doesn't have an infrared (IR) light filter. This means the camera can pick up infrared light that's invisible to humans. This is useful for seeing in seemingly complete darkness--if there's a source of infrared illumination, like from IR LEDs (http://adafru.it/38895983), then the Pi NoIR camera will be able to get a good image.

Here's an example of the NoIR camera image under normal lighting:

Now the same setup with almost no light at all in the room (you can only see the cat's eyes reflecting the camera's red LED!):

And finally the setup with six IR LEDs next to the camera providing illumination:

You can see the Pi NoIR camera is very handy for security cameras and other projects that need to see in the dark!

Field of View

The stock Pi camera has a somewhat narrow field of view. The camera was originally designed for cell phones and similar applications so it has about a 35mm film lens equivalent field of view. Here's an example of the stock camera view with a few objects about two feet in front of the camera:

If the field of view is a too narrow you might consider putting a tiny wide-angle lens adapter in front of the Pi camera. For example this wide-angle cell phone camera adapter (http://adafru.it/jd5) is an inexpensive and easy way to increase the field of view.

You'll need to mount the wide-angle lens in front of the Pi camera lens (don't try to attach it directly to the Pi camera lens!). You can 3D print an enclosure specifically designed to accomodate the wide-angle lens(http://adafru.it/jd7), or just cut a hole in a small cardboard box and build your own simple case.

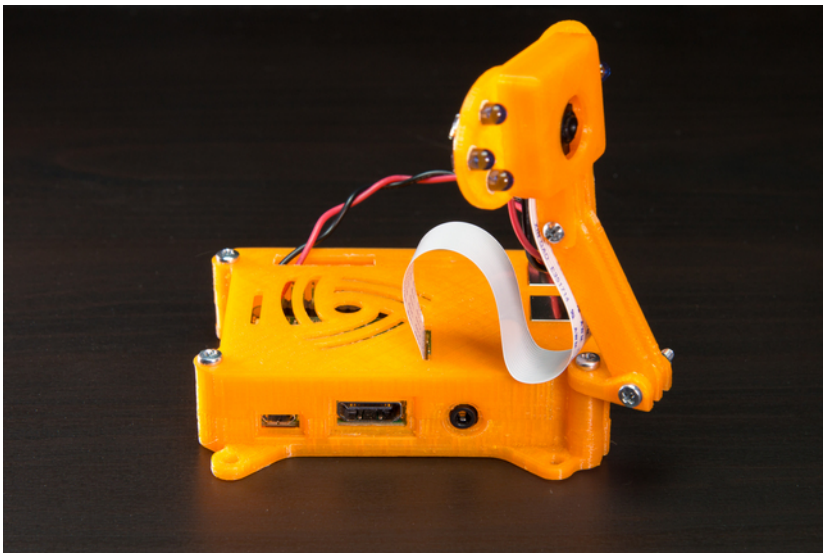
With the wide-angle adapter this is how the same image setup appears:

You'll notice some distortion and vignetting at the edges which is normal for using a wide-angle adapter like this. However the field of view is noticeably larger and better able to view the surrounding room.

Enclosure

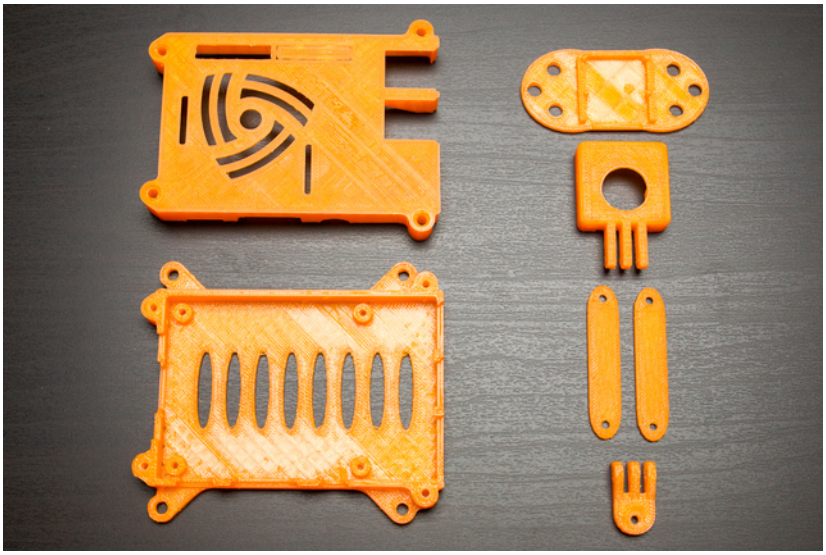
The simplest option for enclosing the project is a small cardboard box. Cut a hole for the camera or tape it to the exterior of the box and feed the cable inside to the Raspberry Pi. If you're using infrared LEDs poke holes in the box and push the LEDs through them. It might not be the prettiest enclosure but it will hold the Pi and camera.

3D Printed Enclosure



You can build a great 3D printed enclosure with an adjustable camera holder by combining parts from a few excellent projects on Thingiverse (http://adafru.it/cor). You'll want to print the following parts:

- Raspberry Pi 2/B+ case with VESA mounts and more(http://adafru.it/jd8)
- Raspberry Pi Camera Case Back for 6x 5mm LEDS (http://adafru.it/jd9)
- Raspberry Pi camera additional parts (http://adafru.it/jda) - You only need the **CameraFrontBottom-fingers.stl** and **fingerclip.stl** files from this project. Note that this camera front can hold most wide-angle lens adapters with a little filing to ensure a snug press fit.
- Raspberry Pi Camera Mount with Ball Joint for Reprap(http://adafru.it/jdb) - You only need to print**two copies** of the**link.stl** file from this project.



You'll need a few screws to assemble the parts too:

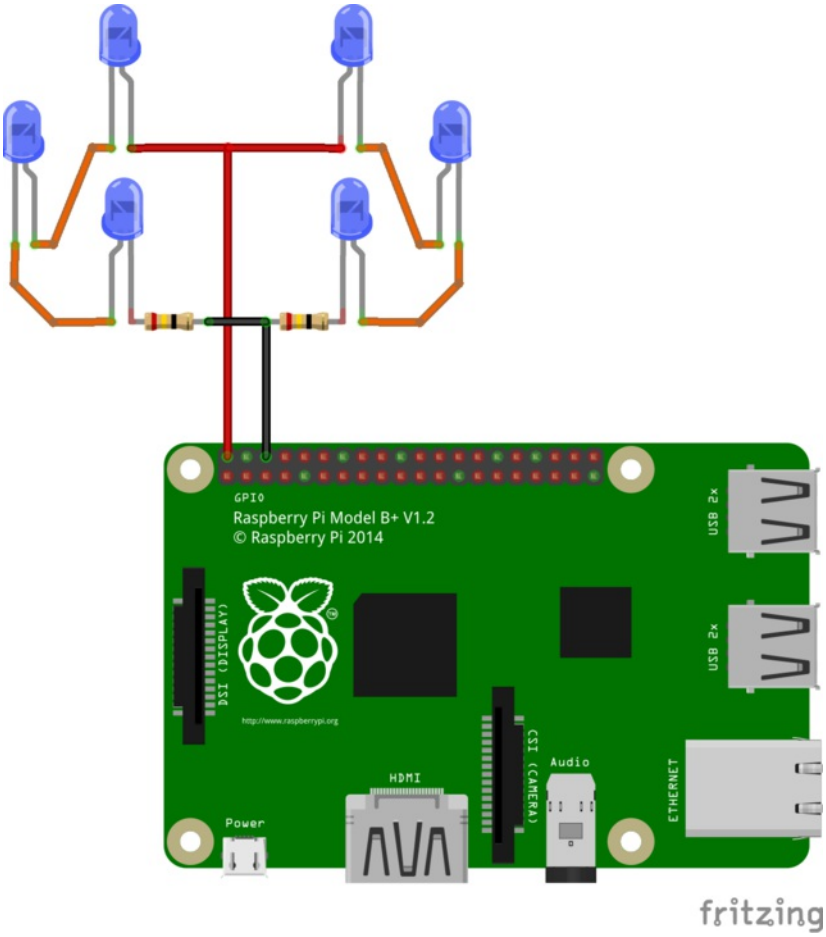
- 4x 4-40 size 3/4" long screws for the Pi case.
- 4x 4-40 size 1/4" long screws for mounting the Pi in the case.
- 2x M3-50 size 12mm or longer screws and nuts for the camera holder.

Check the project pages above for more details on printing and assembling the parts. The parts printed without issue for me on a Printrbot Simple Metal 3D printer (http://adafru.it/1760) using PLA filament. Some filing was necessary to adjust the tolerances and make the camera case parts snap together.

Infrared LED Wiring

If you're using the Pi NoIR filter camera you can add infrared LEDs to provide illumination in the dark. I recommend six high-powered IR LEDs (http://adafru.it/388) placed around the camera. These are just enough LEDs to provide illumination yet still be powered directly from the Raspberry Pi.

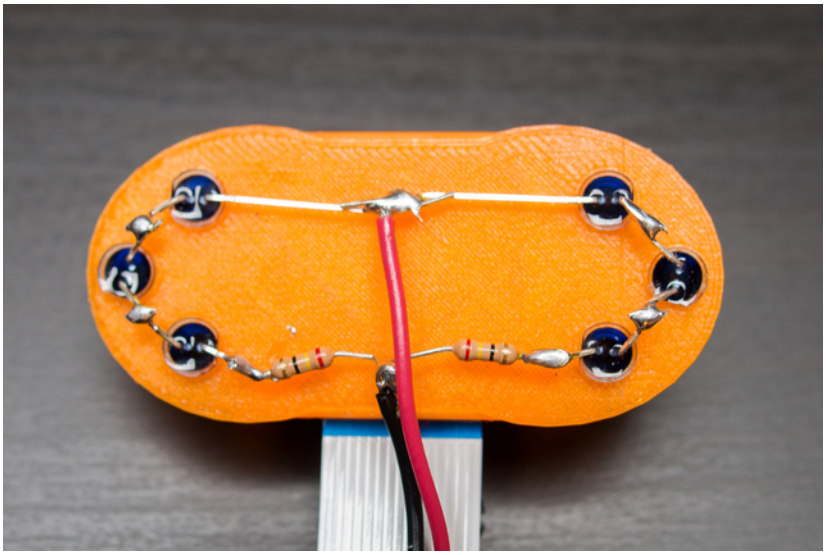
You'll need to solder the LEDs into two parallel sets of three LEDs that are in series like the diagram below:



For each set of LEDs a 24 ohm 1/4 watt resistor will limit the current to around 50mA each (for a total of 100mA, roughly the limit for how much current you want to pull from a Raspberry Pi 5V power line). You can substitute a larger value resistor but the LEDs will be dimmer. **Don't leave out or use smaller value resistors or else you might damage your Pi!**

Be extra careful to make sure you wire the LEDs with the correct polarity so they light as expected. The longer leg of each LED is the anode and is represented by the leg with a crooked/slanted section in the diagram. The shorter leg is the cathode. Make sure to connect the anodes and cathodes of LEDs exactly as shown!

The easiest way to solder the LEDs and resistors is with point-to-point connections. You probably don't need any wires and can just tie the LED legs to each other and solder the connections, then trim with a flush cutter. For example see the wiring below:



The top row with the red wire is connected to the Pi's 5V output (pin 2) and the bottom row with the black wire is connected to a Pi ground pin (pin 6). Use a Pi cobbler (http://adafru.it/914) and breadboard or female connector wires (http://adafru.it/1950) to connect to the Pi pins. See this Pi GPIO pin diagram (http://adafru.it/cbG) if you are unsure exactly which pins are 5V and ground.

Dropbox Sync

Once the Pi camera is setup you can configure the Pi to use the motion software package (http://adafru.it/jdc) to detect movement and upload images to Dropbox (http://adafru.it/jdd). Follow the steps below to first setup sending data to Dropbox with a script, then install and configure motion to use the Dropbox sync script when it detects motion from the camera.

Dropbox is one of many popular consumer cloud storage services. By sending the Pi images to Dropbox you can easily view them from any computer using Dropbox's sync client or its web interface. If you don't have one already you'll need to sign up for a free account with Dropbox (http://adafru.it/aZw) before continuing.

To do the actual image uploading we'll use Andrea Fabrizi's excellent Dropbox Uploader shell script (http://adafru.it/jde). This script can run from a Pi or any other Linux/Unix machine and send data to Dropbox with a simple command. By configuring the motion software to call Dropbox Uploader you'll have a camera uploading images to Dropbox with almost no work.

Note that you can use a different cloud storage service to store camera images. You'll need to make sure there is a script or command that the Pi can run to upload an image to the service. For Dropbox the mentioned Dropbox Uploader script makes the setup and upload process easy, but for other cloud storage services consider:

- OneDrive's Python SDK (http://adafru.it/jdf) - Write a small python script to upload a file to OneDrive's cloud storage service.
- Google Drive uploader script (http://adafru.it/jdg)
- Amazon S3 uploader script (http://adafru.it/jdh)

Dropbox Uploader Setup

Start by installing and configuring the Dropbox Uploader script (http://adafru.it/jde). Make sure the Pi is connected to the Internet and open a terminal on it, then run the following commands to download and install the script:

```
sudo apt-get update
sudo apt-get install -y curl
cd ~
git clone https://github.com/andreafabrizi/Dropbox-Uploader.git
cd Dropbox-Uploader
chmod a+x dropbox_uploader.sh
sudo cp dropbox_uploader.sh /usr/local/bin/dropbox_uploader
```

Once installed you should be able to invoke the dropbox_uploader command to run the script. For example have it print its usage by running the command:

```
dropbox_uploader
```

You should see something like the following displayed:

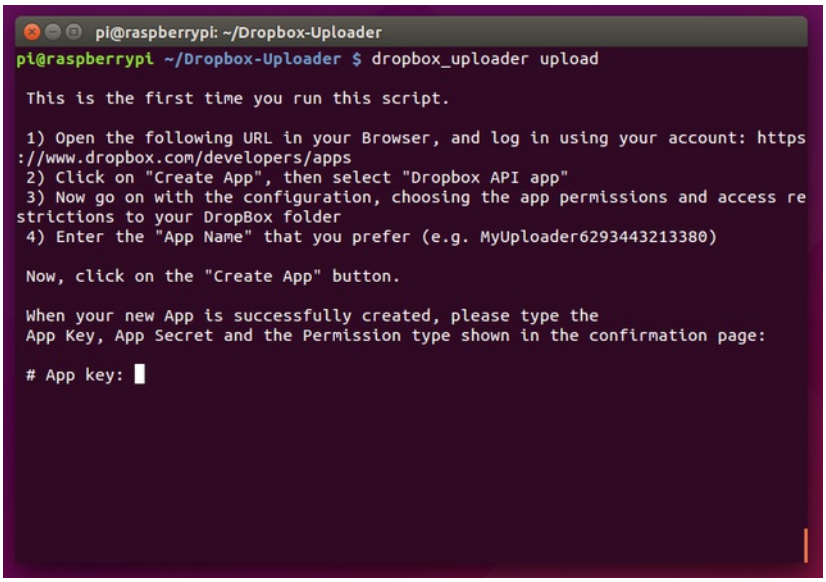
```
Dropbox Uploader v0.16
Andrea Fabrizi - andrea.fabrizi@gmail.com

Usage: /usr/local/bin/dropbox_uploader COMMAND [PARAMETERS]...

Commands:
  upload   <LOCAL_FILE/DIR ...>  <REMOTE_FILE/DIR>
  download <REMOTE_FILE/DIR> [LOCAL_FILE/DIR]
  delete   <REMOTE_FILE/DIR>
  move     <REMOTE_FILE/DIR> <REMOTE_FILE/DIR>
  copy     <REMOTE_FILE/DIR> <REMOTE_FILE/DIR>
  mkdir    <REMOTE_DIR>
  list     [REMOTE_DIR]
  share    <REMOTE_FILE>
  saveurl  <URL> <REMOTE_DIR>
  info
  unlink

Optional parameters:
 -f <FILENAME> Load the configuration file from a specific file
 -s            Skip already existing files when download/upload. Default: Overwrite
 -d            Enable DEBUG mode
 -q            Quiet mode. Don't show messages
 -p            Show cURL progress meter
 -k            Doesn't check for SSL certificates (insecure)

For more info and examples, please see the README file.
```

If you see an error that the dropbox_uploader command is not found carefully check you followed the steps above to install it and try again.

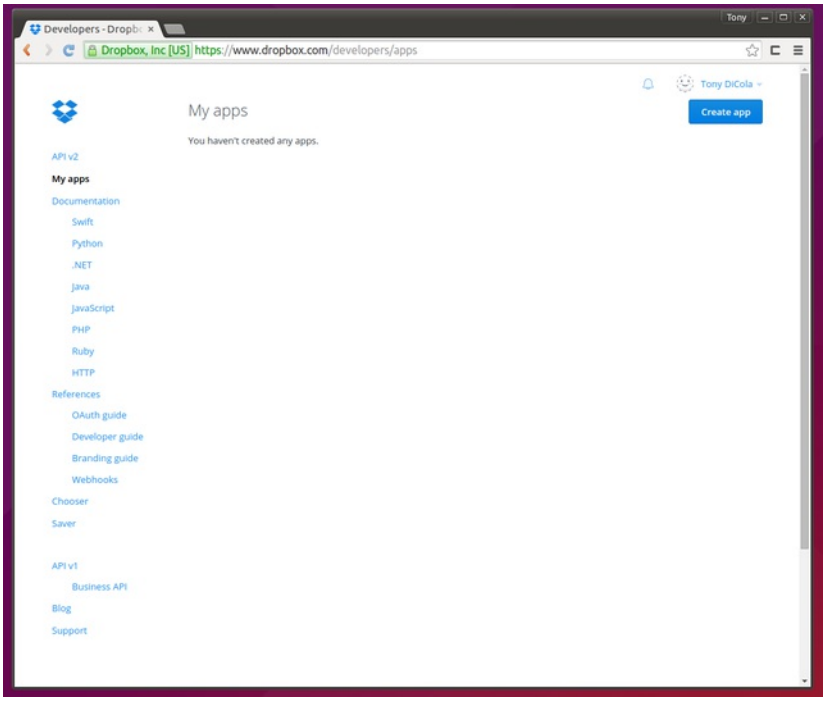
Once the script is installed you'll need to do a one time setup to configure it to talk to your Dropbox account. Start the process by running the upload command:

```
dropbox_uploader upload
```

The Dropbox uploader script will see that it has not been connected to a Dropbox account and walk through the process of setting it up. You should see a screen like the following:



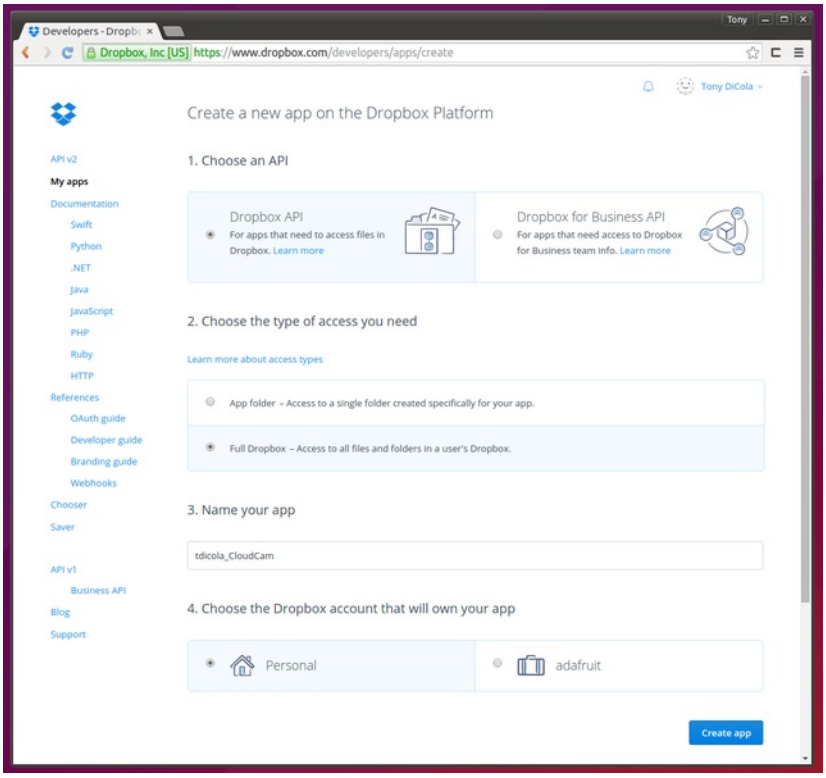
Follow the instructions and open a browser to the[https://www.dropbox.com/developers/apps](http://adafru.it/dfp) URL. Note that you might need to be logged in to Dropbox before you access this URL. The page should look something like the following:



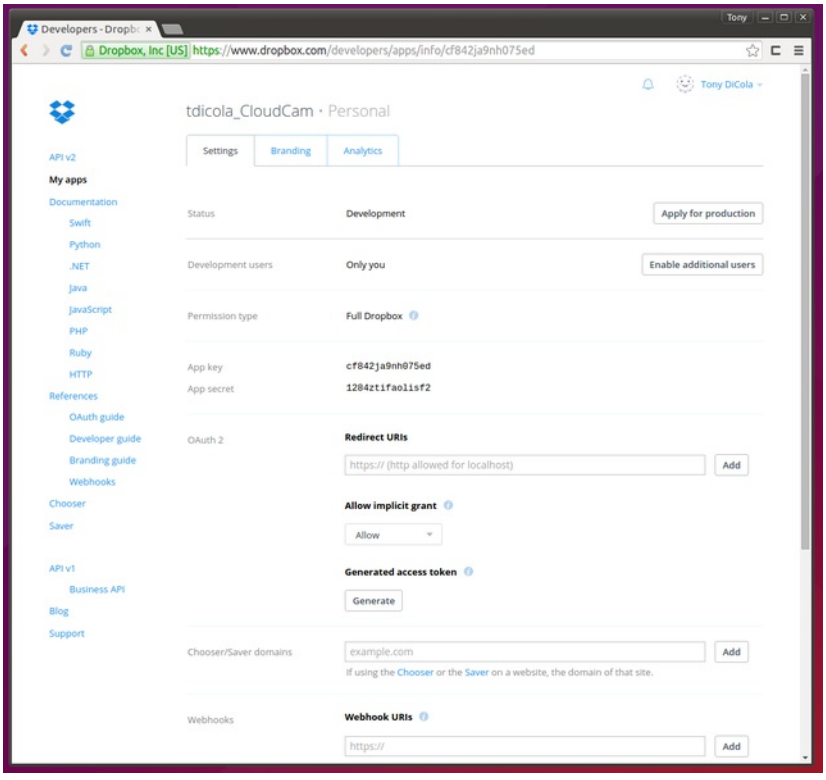
Click the blue **Create app** button in the upper right corner. This should load a page to create a new application that can access your Dropbox account. Fill in the details as follows:

- Under **Choose an API** select **Dropbox API**.
- Under **Choose the type of access you need** select **Full Dropbox** for access to any folder in your Dropbox account.
- For **Name your app** enter a unique name for this application. App names are global across all users so I recommend a name like "(your username)_CloudCam" where "(your username)" is your username. For example I chose "tdicola_CloudCam" for the app name.
- If there's a 4th option to **Choose the Dropbox account that will own your app** select the appropriate owner like your **Personal** account. Don't worry if this option doesn't exist for you, you can skip it and move on.

The page should look something like this for example:



Now click the blue Create app button at the bottom of the page. You should see a new page load that describes details of the application:

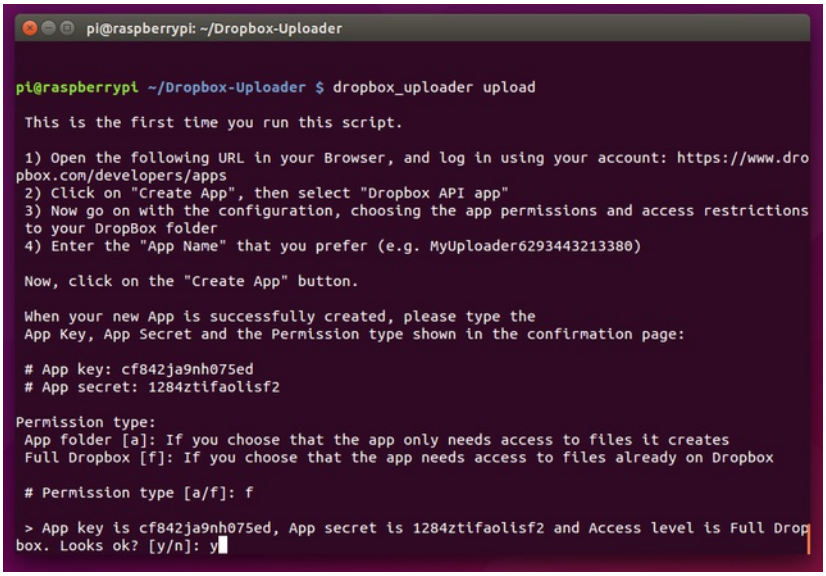


You'll need to copy out a few pieces of information from this page and enter them in the Dropbox uploader script. Specifically you'll want:

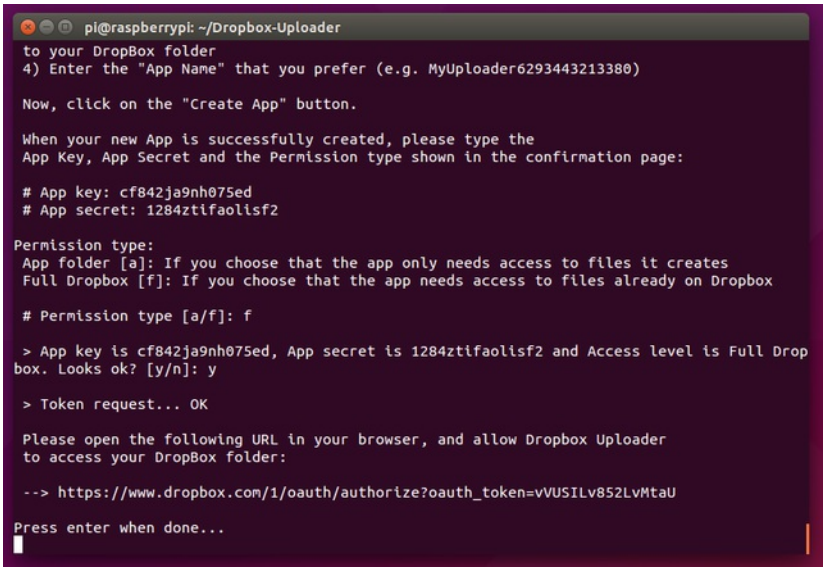
- **App key value**
- **App secret value** (click the show button to see it)

Grab these values and enter them in the Dropbox uploader script's prompts. The script should be asking first for the app key value, then the app secret value, and finally if you set the app to have partial/app permissions or full permissions (you used full permissions if you've followed everything exactly to now).

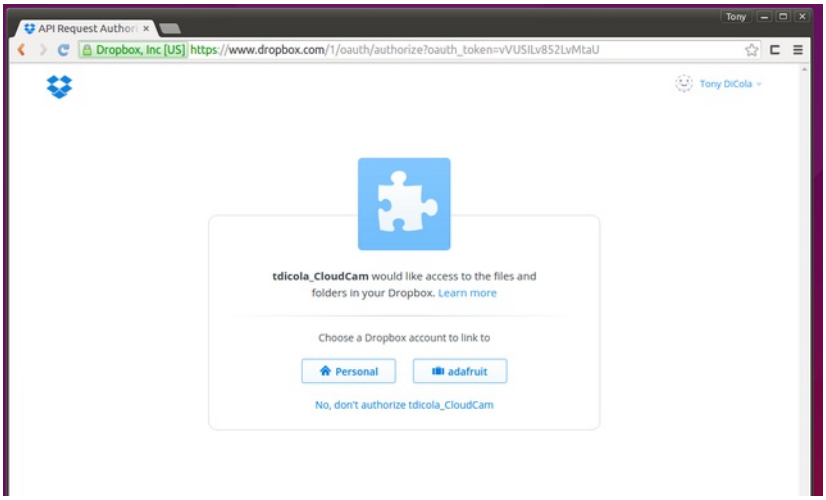
After entering these values you should be at a confirmation prompt like the following:



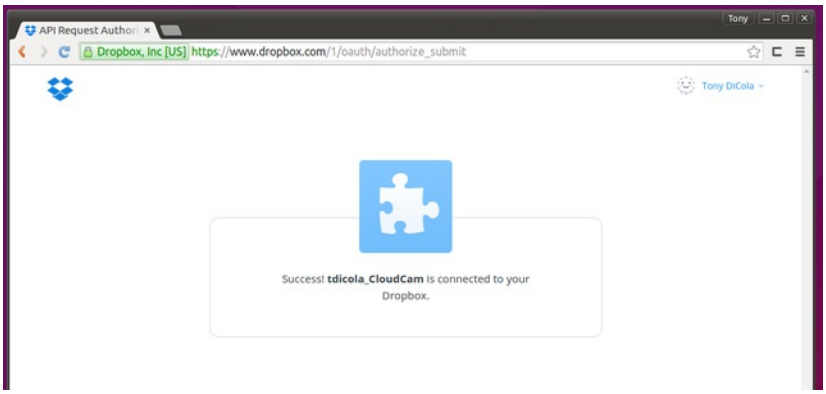
Type **y** and press **enter** to confirm the settings. You should see the script direct you to a new web page similar to the following:



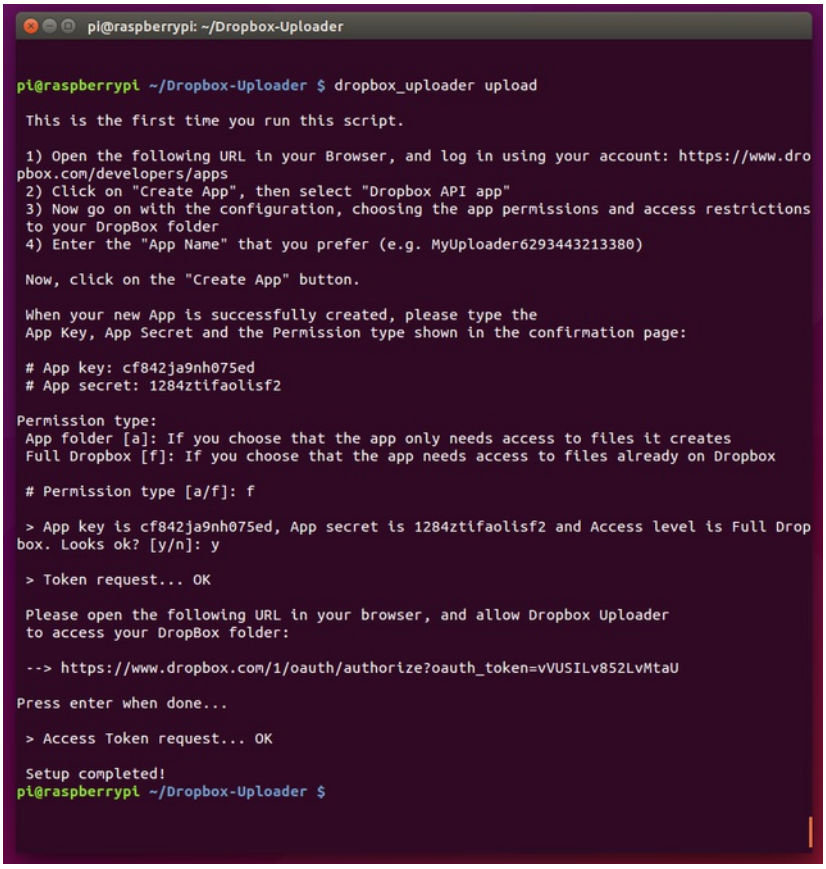
As the script instructs you open a browser and navigate to the URL it provides (make sure you're logged into Dropbox already). You should see a page similar to the following that asks to confirm the app permission request (your page might look different depending on what Dropbox accounts you have setup):



Click the appropriate button to authorize the application to access your Dropbox account. In this case I clicked the **Personal** button to authorize it to access my personal Dropbox account. The page should inform you the application is now connected to Dropbox:



Now go back to the Dropbox uploader script prompt and press **enter** to continue. The script should retrieve an auth token and finish successfully:



If you receive an error go back and carefully check all the application settings, etc. were setup as expected and try again.

Once the script finishes it will save the Dropbox application details in a hidden file under your home directory÷/**.dropbox_uploader** If you ever need to redo the authentication process again just delete this file (**rm ~/.dropbox_uploader**) and run the dropbox_uploader script again.

One important thing you must do is make the **~/.dropbox_uploader** file readable by all users on the Pi. This is necessary because the motion package runs under a different user account and needs to be able to read the Dropbox authentication in the file. Run the following command to make this change:

```
chmod a+r ~/.dropbox_uploader
```

Don't skip running the chmod command above! If you don't make the ~/.dropbox_uploader file readable by all users then motion won't save pictures to Dropbox.

Now that the script is connected to your Dropbox account you can test uploading a document with it. From still inside the Dropbox-Uploader folder try uploading its README.md file to a new Cloud Cam folder on your Dropbox account. Do this by running the command:

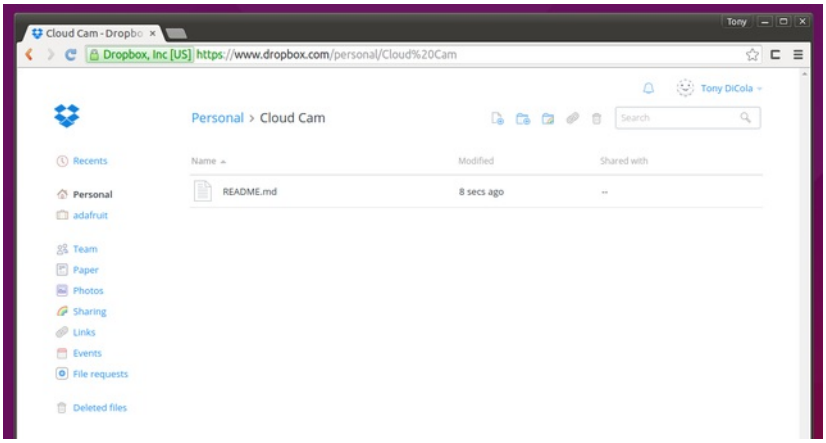
```
dropbox_uploader upload README.md "Cloud Cam/"
```

Make sure to include the trailing slash (/) in the "Cloud Cam/" string! If you forget this slash then the README.md file will be saved to a file called "Cloud Cam" and not a folder!

The script should run and upload the file to your Dropbox:

```
> Uploading "/home/pi/Dropbox-Uploader/README.md" to "/Cloud Cam/README.md"... DONE
```

Open your Dropbox account and look for the Cloud Cam folder. You should see the README.md file there:



If the upload script failed with an error go back and check access to Dropbox has been enabled with the steps above and try again.

Woo hoo! At this point the Pi is ready to start sending data to Dropbox. Now follow the steps below to setup the camera motion detection software and upload data to Dropbox.

Motion Setup

To detect motion with the Pi camera you can use the excellentmotion software package (http://adafru.it/jdc). This program will turn the Pi into a dedicated security camera that can monitor a connected camera to look for motion or periodically capture images.

Be careful to follow the steps below to setup motion on the Raspberry Pi. If you search the internet you might find older instructions for installing a custom build of motion on the Pi. However those instructions won't work with the current Jessie version of Raspbian. You'll need to follow the steps below to load a special Pi camera kernel module, and then you can install and use motion right from the Raspbian package repository.

Pi Camera V4L2 Kernel Module

Before you can use the motion package you'll need to load aspecial kernel module (http://adafru.it/jdi) that will make it work with the Pi camera. Normally the Pi camera talks directly to the Pi's GPU so programs have to be written to specifically use the Pi camera--i.e. the camera doesn't appear like a webcam or other video source. However the Pi foundation created a special kernel module to make the Pi camera work with Linux's Video4Linux 2 API and look like a normal video source. Using the Pi camera V4L2 module you can use the Pi camera with motion and most other Linux video programs.

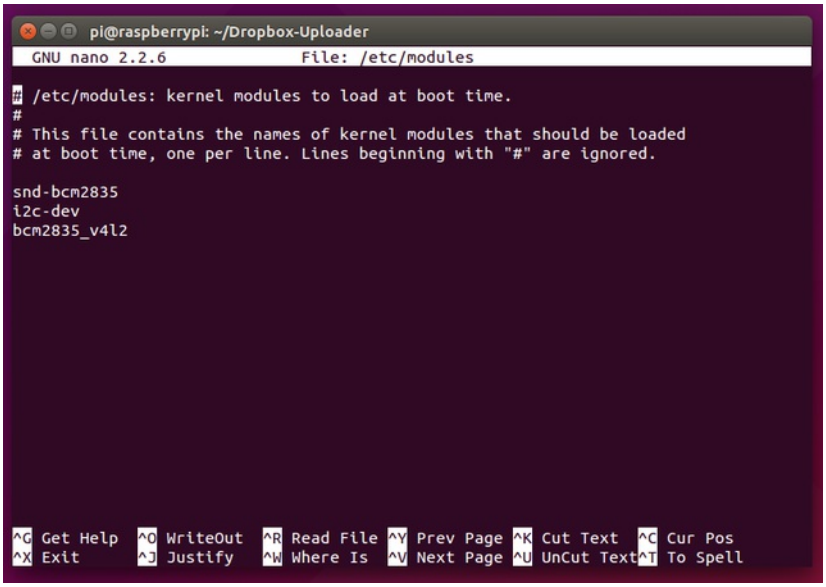
To load this module first make sure you're using the latest Raspbian Jessie image on the Pi. Then connect to the Pi and run the following command to edit the /etc/modules configuration file:

```
sudo nano /etc/modules
```

This file controls what extra kernel modules are loaded on boot and we need to add a new line to include the special Pi camera V4L2 module. Scroll down to the bottom of the file and add the following line:

```
bcm2835_v4l2
```

There might be other lines in the file so leave them alone and add the bcm2835_v4l2 line at the end of the file. For example here's how a modified configuration file should look:



Save the file and exit the editor by pressing **Ctrl-o** then **enter** and then **Ctrl-x**.

Now reboot the Pi by running:

```
sudo reboot
```

After the Pi boots again connect in a SSH terminal and run the following command to check the module successfully loaded and created a video source for the Pi camera:

```
ls -l /dev/video*
```

You should see a **/dev/video0** source listed, like:

```
crw-rw----+ 1 root video 81, 0 Nov  4 04:46 /dev/video0
```

If you don't see the **/dev/video0** source you can run the **lsmod** command to list all the active kernel modules and check if the bcm2835_v4l2 modules is loaded. You can attempt to manually load the module by running **sudo modprobe bcm2835_v4l2**. If all else fails check the kernel log by running the **dmesg** command to see if there are any error messages which might indicate a problem loading the module. The Pi camera V4L thread on the Raspberry Pi forums (http://adafru.it/jdi) might be able to troubleshoot and provide more insight into problems loading the module.

Once you've confirmed the Pi camera V4L kernel module is loaded and a **/dev/video0** device exists move on to the next section to setup the motion package.

Motion Install & Configuration

With the Pi camera configured as a Linux video source you can now use it with software like motion. First you'll want to install motion by running the following command:

```
sudo apt-get update
sudo apt-get install -y motion
```

You should see text similar to the following as motion is installed:

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  mysql-client postgresql-client
Recommended packages:
  ffmpeg
The following NEW packages will be installed:
  motion
0 upgraded, 1 newly installed, 0 to remove and 15 not upgraded.
Need to get 0 B/230 kB of archives.
After this operation, 746 kB of additional disk space will be used.
Preconfiguring packages ...
Selecting previously unselected package motion.
(Reading database ... 123293 files and directories currently installed.)
Preparing to unpack .../motion_3.2.12+git20140228-4+b2_armhf.deb ...
Unpacking motion (3.2.12+git20140228-4+b2) ...
Processing triggers for man-db (2.7.0.2-5) ...
```

```
Processing triggers for systemd (215-17+deb8u2) ...
Setting up motion (3.2.12+git20140228-4+b2) ...
```

After the package installs you'll want to run a command to fix an issue with the motion package not setting the right ownership for the location where it stores images:

```
sudo mkdir /var/lib/motion
sudo chown motion:motion /var/lib/motion
```

Now you'll need to edit the global configuration file for motion to customize its behavior. Firstskim this page on motion's configuration options (http://adafru.it/jdj) to get an overview of the available options. Then run the following command to start editing the /etc/motion/montion.conf file:

```
sudo nano /etc/motion/motion.conf
```

Most options you'll want to leave the same as the default, but scroll down to these lines that control the size of the captured image:

```
# Image width (pixels). Valid range: Camera dependent, default: 352
width 320

# Image height (pixels). Valid range: Camera dependent, default: 288
height 240
```

Bump the size of the captured image up to 1280x720 pixels (720p) by making those lines look like:

```
# Image width (pixels). Valid range: Camera dependent, default: 352
width 1280

# Image height (pixels). Valid range: Camera dependent, default: 288
height 720
```

Another option to change is the motion threshold. This setting controls how many pixels have to change between frames before motion is detected. The default value is 1500 pixels, but that's a relatively small value for a 720p frame (which has almost 1 million pixels). Increase the value to 3000 to start:

```
# Threshold for number of changed pixels in an image that
# triggers motion detection (default: 1500)
threshold 3000
```

If the camera is too sensitive you can increase this threshold to a larger value, or if the camera isn't sensitive enough try droppping it to a lower value. You might need to experiment with different values to see what works best for your setup.

The minimum_motion_frames setting is another useful setting to help control the sensitivity of motion detection. The default setting looks like:

```
# Picture frames must contain motion at least the specified number of frames
# in a row before they are detected as true motion. At the default of 1, all
# motion is detected. Valid range: 1 to thousands, recommended 1-5
minimum_motion_frames 1
```

You can increase this value to require more than 1 consecutive changed frame before triggering motion. For example if you set this to 3 then there must be 3 consecutive frames that differ by the threshold amount of pixels before motion is detected.

This setting can help deal with false positives where motion is detected from random camera noise, dust, etc. The trade off is that very fast motion that only occurs for a frame or two won't be detected.

I like setting the value to 2 so that two consecutive frames must have motion before the motion event is triggered. This prevents a random blip or dust particle from triggering motion, but should still pick up most quick events. Change the value to look like:

```
# Picture frames must contain motion at least the specified number of frames
# in a row before they are detected as true motion. At the default of 1, all
# motion is detected. Valid range: 1 to thousands, recommended 1-5
minimum_motion_frames 2
```

Another setting to change is the video capture setting. Scroll down to this part of the configuration:

```
# Use ffmpeg to encode movies in realtime (default: off)
ffmpeg_output_movies on
```

To keep this project simple we'll disable video output--change the setting to off:

```
# Use ffmpeg to encode movies in realtime (default: off)
ffmpeg_output_movies off
```

Feel free to explore enabling movies later, but be aware it might require other software to be installed or use a lot of the Pi's CPU (particularly if the resolution is high like 720p or 1080p).

The snapshot_interval setting allows you to have the camera take a photo at a certain frequency regardless of there being motion or not:

```
# Make automated snapshot every N seconds (default: 0 = disabled)
snapshot_interval 0
```

You probably don't want to turn this setting on because it can generate a lot of data and overload your Dropbox account, however it's good to know that it exists and is an option if you don't need motion detection.

The target_dir setting controls where captured images are stored locally on the Pi. You don't need to change this setting, but it is good to know that a copy of captured images will be stored in this location (the default is /var/lib/motion):

```
# Target base directory for pictures and films
# Recommended to use absolute path. (Default: current working directory)
target_dir /var/lib/motion
```

You can enable a video stream of the camera by changing the stream_localhost setting:

```
# Restrict stream connections to localhost only (default: on)
stream_localhost on
```

By setting stream_localhost to off then any computer on your network can view a video stream from the Pi's camera. This is useful for setting up the camera and making sure it has a good view of what you want to capture. You can enable the stream by setting the value to off:

```
# Restrict stream connections to localhost only (default: on)
stream_localhost off
```

Be aware that anyone on your network can view the stream! By default there is no authentication or other login required to see the stream.

Finally the last important setting and one that you must change is the on_picture_save setting which controls what action happens when the camera takes a picture (either from detecting motion or as part of a periodic picture capture):

```
# Command to be executed when a picture (.ppm|.jpg) is saved (default: none)
# To give the filename as an argument to a command append it with %f
; on_picture_save value
```

By default there is no action defined for this setting and it is commented out with a semicolon. To enable the sync of photos to Dropbox we can insert a call to the Dropbox uploader script (that was setup in the previous section).

Modify the on_picture_save configuration to look exactly like the following:

```
# Command to be executed when a picture (.ppm|.jpg) is saved (default: none)
# To give the filename as an argument to a command append it with %f
on_picture_save dropbox_uploader -f /home/pi/.dropbox_uploader upload %f "Cloud Cam/"
```

This will tell motion to call the dropbox_uploader command and upload the picture to the "Cloud Cam" folder on Dropbox. Notice the -f parameter is used to point to the configuration file that was created during the Dropbox app setup (which lives as a hidden file in the pi user's home directory).

That's all you need to change to setup motion! Save the configuration file and exit the text editor by pressing **Ctrl-o** then **enter** and then **Ctrl-x**.

Now test that motion runs and uploads pictures to Dropbox. Manually run motion by executing:

```
sudo motion -n
```

You should see motion start and print some initialization that looks similar to:

```
[0] [NTC] [ALL] conf_load: Processing thread 0 - config file /etc/motion/motion.conf
[0] [ALR] [ALL] conf_cmdparse: Unknown config option "sdl_threadnr"
[0] [NTC] [ALL] motion_startup: Motion 3.2.12+git20140228 Started
[0] [NTC] [ALL] motion_startup: Logging to syslog
[0] [NTC] [ALL] motion_startup: Using log type (ALL) log level (NTC)
[0] [NTC] [ENC] ffmpeg_init: ffmpeg LIBAVCODEC_BUILD 3670016 LIBAVFORMAT_BUILD 3670272
[0] [NTC] [ALL] main: Thread 1 is from /etc/motion/motion.conf
[0] [NTC] [ALL] main: Thread 1 is device: /dev/video0 input -1
[0] [NTC] [ALL] main: Stream port 8081
[0] [NTC] [ALL] main: Waiting for threads to finish, pid: 1875
[1] [NTC] [ALL] motion_init: Thread 1 started , motion detection Enabled
[1] [NTC] [VID] vid_v4lx_start: Using videodevice /dev/video0 and input -1
[1] [NTC] [VID] v4l2_get_capability:
------------------------
cap.driver: "bm2835 mmal"
cap.card: "mmal service 16.1"
cap.bus_info: "platform:bcm2835-v4l2"
cap.capabilities=0x85200005
------------------------
[1] [NTC] [VID] v4l2_get_capability: - VIDEO_CAPTURE
[1] [NTC] [VID] v4l2_get_capability: - VIDEO_OVERLAY
[1] [NTC] [VID] v4l2_get_capability: - READWRITE
[1] [NTC] [VID] v4l2_get_capability: - STREAMING
[1] [NTC] [VID] v4l2_select_input: name = "Camera 0", type 0x00000002, status 00000000
[1] [NTC] [VID] v4l2_select_input: - CAMERA
[0] [NTC] [STR] httpd_run: motion-httpd testing : IPV4 addr: 127.0.0.1 port: 8080
```

```
[1] [WRN] [VID] v4l2_select_input: Device doesn't support VIDIOC_G_STD
[0] [NTC] [STR] httpd_run: motion-httpd Bound : IPV4 addr: 127.0.0.1 port: 8080
[1] [NTC] [VID] v4l2_do_set_pix_format: Testing palette YU12 (1280x720)
[0] [NTC] [STR] httpd_run: motion-httpd/3.2.12+git20140228 running, accepting connections
[1] [NTC] [VID] v4l2_do_set_pix_format: Using palette YU12 (1280x720) bytesperlines 1280 sizeimage 1382400 colorspace 00000001
[0] [NTC] [STR] httpd_run: motion-httpd: waiting for data on 127.0.0.1 port TCP 8080
[1] [NTC] [VID] v4l2_scan_controls: found control 0x00980900, "Brightness", range 0,100
[1] [NTC] [VID] v4l2_scan_controls:  "Brightness", default 50, current 50
[1] [NTC] [VID] v4l2_scan_controls: found control 0x00980901, "Contrast", range -100,100
[1] [NTC] [VID] v4l2_scan_controls:  "Contrast", default 0, current 0
[1] [NTC] [VID] v4l2_scan_controls: found control 0x00980902, "Saturation", range -100,100
[1] [NTC] [VID] v4l2_scan_controls:  "Saturation", default 0, current 0
[1] [NTC] [VID] v4l2_scan_controls: found control 0x0098090e, "Red Balance", range 1,7999
[1] [NTC] [VID] v4l2_scan_controls:  "Red Balance", default 1000, current 1000
[1] [NTC] [VID] v4l2_scan_controls: found control 0x0098090f, "Blue Balance", range 1,7999
[1] [NTC] [VID] v4l2_scan_controls:  "Blue Balance", default 1000, current 1000
[1] [NTC] [VID] vid_v4lx_start: Using V4L2
[1] [NTC] [ALL] image_ring_resize: Resizing pre_capture buffer to 1 items
[1] [NTC] [STR] http_bindsock: motion-stream testing : IPV4 addr: 0.0.0.0 port: 8081
[1] [NTC] [STR] http_bindsock: motion-stream Bound : IPV4 addr: 0.0.0.0 port: 8081
[1] [NTC] [ALL] motion_init: Started motion-stream server in port 8081 auth Disabled
[1] [NTC] [ALL] image_ring_resize: Resizing pre_capture buffer to 2 items
```

If you see an error go back and carefully check that motion was installed, and that the configuration was updated as described above and try again.

Now move something in front of the camera. After a few moments you should see a motion event start:

```
[1] [NTC] [ALL] motion_detected: Motion detected - starting event 1
[1] [NTC] [EVT] event_newfile: File of type 1 saved to: /var/lib/motion/01-20151104082928-00.jpg
[1] [NTC] [EVT] event_newfile: File of type 1 saved to: /var/lib/motion/01-20151104082928-01.jpg
 > Uploading "/var/lib/motion/01-20151104082928-00.jpg" to "/Cloud Cam/01-20151104082928-00.jpg"...  > Uploading "/var/lib/motion/01-20151104082928-01.jpg" to "/Cloud Cam/0
DONE
```

Notice the lines that look like " > Uploading "/var/lib/motion/01-20151104082928-00.jpg" to "/Cloud Cam/01-20151104082928-00.jpg"... DONE", that means the dropbox_uploader script successfully uploaded the image to Dropbox. If you open your Dropbox Cloud Cam folder you should see the image file is now there--woo hoo!

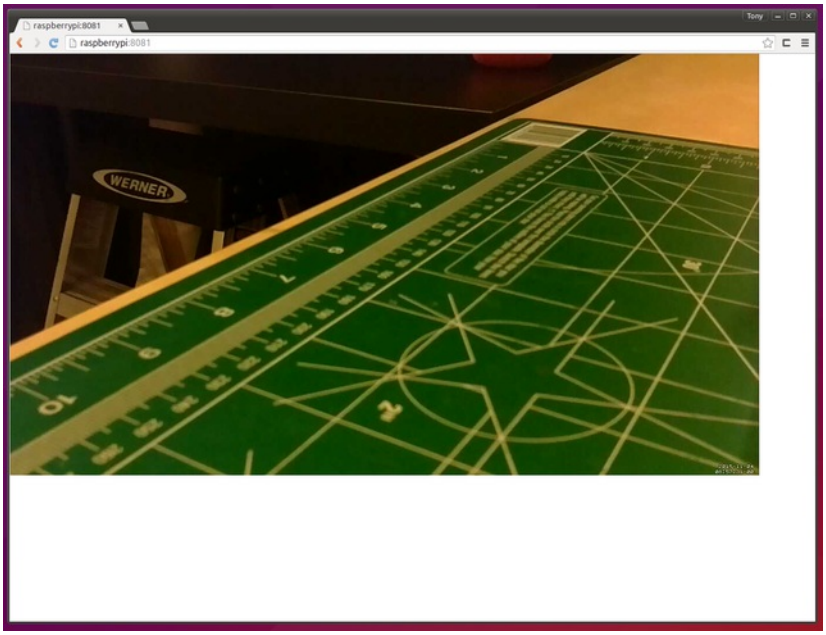
If you see an error or don't see the image in Dropbox go back and check the dropbox_uploader script was installed and setup as described above. Make sure you can manually use the script to upload to your Dropbox account. Also be certain that you ran the chmod a+r command to make the ~/.dropbox_uploader file readable by all users as described above.

If you're curious about the meaning of the file names that motion writes, for example '01-20151104082928-00.jpg', the file name is composed of these parts:

- **Event number** - This is a numer that increases every time there is a new motion event since the motion program started running. This is the value 01 in the example above.
- **Dash**
- **Date** - The date in year, month, day format. The example above is November 4th, 2015.
- **Time** - The time in hour, minute, second format. The example above is 8:29:28 AM.
- **Dash**
- **Frame number** - This is just a numeric ID of the frame within this motion event. Larger values are further in the future than smaller values. The example above is a frame number 00.

You can actually change this file format by modifying the jpeg_filename option in motion's config file.

One final thing you can check with motion running is the output of its video stream. From a web browser access http://raspberrypi:8081/ (http://adafru.it/jdk) (note that you might need to substitute raspberrypi in the URL with the IP address of your Raspberry Pi). You should see the video from the camera, for example:



Now stop motion by pressing **Ctrl-c** in the terminal. After a few moments it will terminate and return you to the console. In the next section you'll configure motion to automatically start on boot.

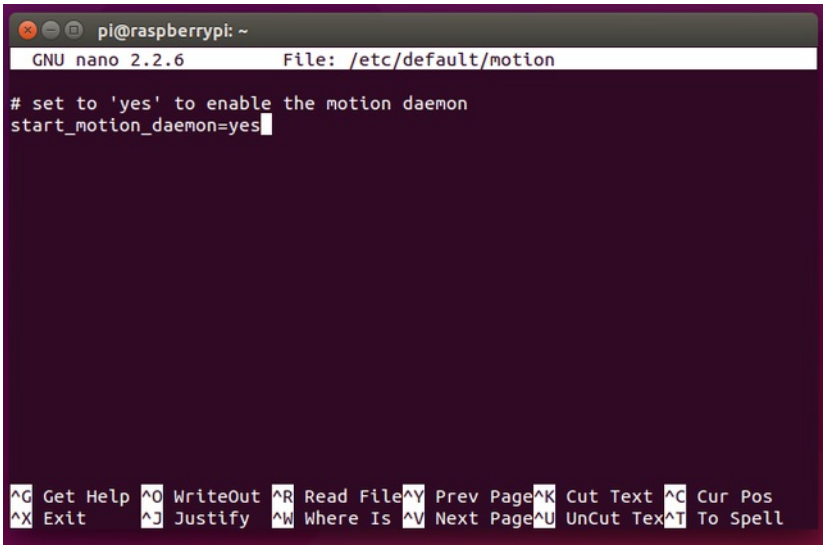
Run Motion on Boot

To configure motion to run on boot you'll need to run a few commands that enable its init script.

First edit the /etc/default/motion file to enable its daemon mode by running:

```
sudo nano /etc/default/motion
```

Change the **start_motion_daemon=no** line to **yes**. The file should look like the following:



Save and exit by pressing **Ctrl-o** then **enter** and then **Ctrl-x**.

Now enable the motion service with Raspbian Jessie's systemd service by running:

```
sudo systemctl enable motion
```

You should see this command print information about synchronizing the state of the motion.service:

```
Synchronizing state for motion.service with sysvinit using update-rc.d...
Executing /usr/sbin/update-rc.d motion defaults
Executing /usr/sbin/update-rc.d motion enable
```

Now reboot the Pi and check that the red camera light turns on to show that motion is running. You should also be able to move in front of the camera and see images uploaded to Dropbox. You can also connect to motion's video stream in a browser to check what the camera is seeing.

Finally connect to the Pi in a terminal again to see some commands that control and troubleshoot motion. First you can see motion's log using systemd's journalctl tool:

```
sudo journalctl -u motion
```

After running the command all of the output of motion will be printed. Press down or page down to page through results, and q to quit. If you have problems or motion doesn't run check the log file to see if there is an error that can help fix the problem.

You can also check the status of the motion service by running:

```
sudo systemctl status motion
```

This will print out information about the motion process and is useful to check if it's running (look for the 'active (running)' status).

You can stop motion by running:

```
sudo systemctl stop motion
```

Note that this will only stop motion until the next boot. If you want to permanently disable motion so that it doesn't run at boot anymore run:

```
sudo systemctl disable motion
```

If you're curious you can use other systemd commands to manipulate the motion service. You can learn more about systemd's commands from this great Arch Linux systemd wiki page (http://adafru.it/jdl) (even though it's for a different Linux distribution the information still pertains to Raspbian Jessie).

That's all there is to running the motion package and uploading images to Dropbox with the cloud cam!

Adafruit IO

You can setup the cloud cam project to send to feeds onAdafruit IO (http://adafru.it/fH9). Using these image feeds you can build interesting dashboards that combine live camera views, sensor readings, and more.

First you'll need to have access to the Adafruit IO beta--right now it's a limited invite beta but will be expanding more over time.

Next it will help to familiarize yourself with the following guides to get an overview of Adafruit IO:

- Adafruit IO Basics Series (http://adafru.it/iDX) (in particular the feeds and dashboard guides)
- Adafruit IO Overview (http://adafru.it/jdm)

You can also check out some example projects that use Adafruit IO to help understand its capbilities and get some inspiration:

- Track Your Treats GPS Candy Tracker (http://adafru.it/jdn)
- A Sillier Mousetrap: Logging Mouse Data to Adafruit IO (http://adafru.it/iRA)

Follow the steps below to setup the Pi to send data to Adafruit IO.**Make sure you're using the Raspbian Jessie operating system release and have followed the Pi Camera Setup steps (http://adafru.it/jdo) before continuing!**

Install Node.js

First you'll need to install the latest stable version of Node.js for the Raspberry Pi. With the recent Raspbian Jessie release this process is much easier than it was in the past. You just need to download a pre-built Node.js package from the node-arm project (http://adafru.it/ehD) and install it.

Be aware that you'll need to make sure you don't already have a version of Node.js installed on the Pi. If you're unsure or have an old version installed I recommend starting with a fresh new Raspbian Jessie image.

In a command terminal on the Pi execute the following commands:

```
cd ~
wget http://node-arm.herokuapp.com/node_latest_armhf.deb
sudo dpkg -i node_latest_armhf.deb
```

This should install a recent stable version of Node.js and output text like:

```
Selecting previously unselected package node.
(Reading database ... 117337 files and directories currently installed.)
Preparing to unpack node_latest_armhf.deb ...
Unpacking node (4.2.1-1) ...
Setting up node (4.2.1-1) ...
Processing triggers for man-db (2.7.0.2-5) ...
```

To double check Node.js and the NPM package manager are installed you can run commands to check their version.

For example to check Node.js run:

```
node -v
```

At the time of writing this guide the current stable version is:

```
v4.2.1
```

And to check NPM run:

```
npm -v
```

Which at the time of writing was version:

```
2.14.7
```

Install adafruit-io-camera

Once the latest Node.js version is installed you can move on to intall the adafruit-io-camera tool. This tool will monitor the Pi camera and send pictures to an Adafruit IO feed where they can be displayed on a dashboard.

To install adafruit-io-camera run:

```
sudo apt-get update
sudo apt-get install -y imagemagick
```

```
sudo npm install --global --no-optional forever forever-service adafruit-io-camera
```

If you're interested in looking at the source code for the library, you can find it at https://github.com/adafruit/adafruit-io-camera (http://adafru.it/xxB).

You should see NPM go through and install the adafruit-io-camera tool and all of its dependencies:

```
/usr/local/bin/forever-service -> /usr/local/lib/node_modules/forever-service/bin/forever-service
/usr/local/bin/get-forever-config -> /usr/local/lib/node_modules/forever-service/bin/get-forever-config
npm WARN engine express-csv@0.6.0: wanted: {"node":"0.6 || 0.8 || 0.10"} (current: {"node":"4.2.1","npm":"2.14.7"})
/usr/local/bin/forever -> /usr/local/lib/node_modules/forever/bin/forever
/usr/local/bin/adafruit-io -> /usr/local/lib/node_modules/adafruit-io-camera/cli.js
forever-service@0.5.4 /usr/local/lib/node_modules/forever-service
├── commander@2.3.0
├── async@0.9.2
├── shelljs@0.3.0
├── walker@1.0.7 (makeerror@1.0.11)
└── swig@1.4.2 (optimist@0.6.1, uglify-js@2.4.24)

forever@0.15.1 /usr/local/lib/node_modules/forever
├── path-is-absolute@1.0.0
├── object-assign@3.0.0
├── clone@1.0.2
├── colors@0.6.2
├── timespan@2.3.0
├── optimist@0.6.1 (wordwrap@0.0.3, minimist@0.0.10)
├── nssocket@0.5.3 (eventemitter2@0.4.14, lazy@1.0.11)
├── cliff@0.1.10 (eyes@0.1.8, colors@1.0.3)
├── prettyjson@1.1.3 (colors@1.1.2, minimist@1.2.0)
├── winston@0.8.3 (cycle@1.0.3, async@0.2.10, eyes@0.1.8, stack-trace@0.0.9, isstream@0.1.2, pkginfo@0.3.1)
├── utile@0.2.1 (async@0.2.10, deep-equal@1.0.1, ncp@0.4.2, i@0.3.3, mkdirp@0.5.1, rimraf@2.4.3)
├── shush@1.0.0 (strip-json-comments@0.1.3, caller@0.0.1)
├── nconf@0.6.9 (ini@1.3.4, async@0.2.9, optimist@0.6.0)
├── forever-monitor@1.6.0 (minimatch@2.0.10, ps-tree@0.0.3, chokidar@1.2.0, broadway@0.3.6)
└── flatiron@0.4.3 (director@1.2.7, optimist@0.6.0, prompt@0.2.14, broadway@0.3.6)

adafruit-io-camera@1.0.2 /usr/local/lib/node_modules/adafruit-io-camera
├── dotenv@1.2.0
├── motion@0.2.2 (im-decode@0.1.2, ttq@0.1.2)
├── yargs@3.29.0 (decamelize@1.1.1, camelcase@1.2.1, window-size@0.1.2, y18n@3.2.0, os-locale@1.4.0, cliui@3.0.3)
├── raspicam@0.2.12 (lodash@3.10.1)
└── adafruit-io@4.2.5 (express-csv@0.6.0, xml@1.0.0, chalk@1.1.1, mkdirp@0.5.1, nedb@1.2.1, express@4.13.3, winston@1.1.2, mqtt@1.5.0, inquirer@0.10.1, restler@3.4.0, sw
```

Configure adafruit-io-camera

After installing the tool you'll need to configure adafruit-io-camera to access your Adafruit IO account. Before you start make sure you have the following information:

- **Adafruit account username** - Find this by logging in to https://accounts.adafruit.com/ (http://adafru.it/dyy) and copying the **Username** field value.
- **Adafruit IO key** - Find this by accessing https://io.adafruit.com/settings (http://adafru.it/fVT) and clicking the yellow **View AIO Keys** button, then copying the key value out (it will be a long string of hex characters).

Now to configure the camera run the following command:

```
adafruit-io camera config --username USERNAME --key KEY
```

Where **USERNAME** is your Adafruit account username and **KEY** is your Adafruit IO key value. For example if your username were **mosfet** and and you had a key value of **0123ffff** you would run:

```
adafruit-io camera config --username mosfet --key 0123ffff
```

Start adafruit-io-camera

Once the camera tool is configured you can turn it on to start sending picture data to an Adafruit IO feed. Right now the easiest way to use the camera tool is to have it upload a new image periodically, like once every two seconds. This way you can build a dashboard that shows camera feeds, sensor data, and buttons to interact with devices.

To start the camera and have it send an image to a feed every two seconds run the following command:

```
adafruit-io camera start -f camera_feed -m false -r 2
```

This parameters to the command have the following meaning:

- **start** - This is the camera command that tells the tool to start the camera process.
- **-f camera_feed** - This tells the tool to write image data to the feed named **camera_feed**. You can change this value to write to a different

feed. Don't worry if the feed doesn't exist as it will be created automatically.

- **-m false** - This explicitly turns off motion detection and instead sends a new image to the camera feed at a fixed frequency. Motion detection support in the tool and Adafruit IO dashboard is still a bit new and under development. You can experiment with turning on motion detection, but be aware only the most recently detected motion image will be stored in the feed (i.e. you can't see a history or log of motion events yet).
- **-r 2** - This tells the tool to write a new image to the feed every 2 seconds. You can use a different value (in seconds) to change how often image data is written. Remember though you can only write data about once a second to Adafruit IO so keep this to a value like once every 2 seconds at most.

Once the camera tool loads you should see it print something like the following and exit:

```
--------------------------------------------------------------------------------
                                adafruit io
--------------------------------------------------------------------------------
[info] starting camera...
[info] camera daemon started and is pushing images to Adafruit IO
```

If you see an error go back and check the camera tool was configured to use your Adafruit account username and IO key and try again.

Note that it takes a few seconds for the camera process to start and the red camera LED to turn on.

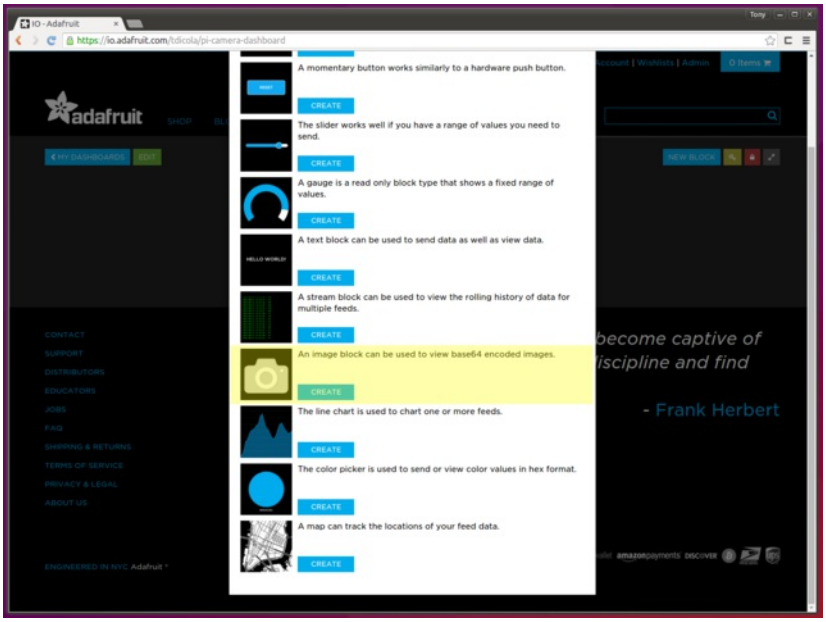
Create Dashboard

With the camera tool started and sending data to a feed you're ready to make a dashboard to view the data. If you're new to creating dashboards it will help to familiarize yourself with a few guides that walk through the process of creating a dashboard:

- Adafruit IO Basics: Dashboards (http://adafru.it/f5m)
- Datalogging Hat Dashboard Creation (http://adafru.it/jdp)

To start navigate to https://io.adafruit.com/dashboards (http://adafru.it/eIS) while you're logged in to your Adafruit account. Click the **Create Dashboard** button in the upper right corner and give the new dashboard a descriptive name like **Pi Camera Dashboard**.

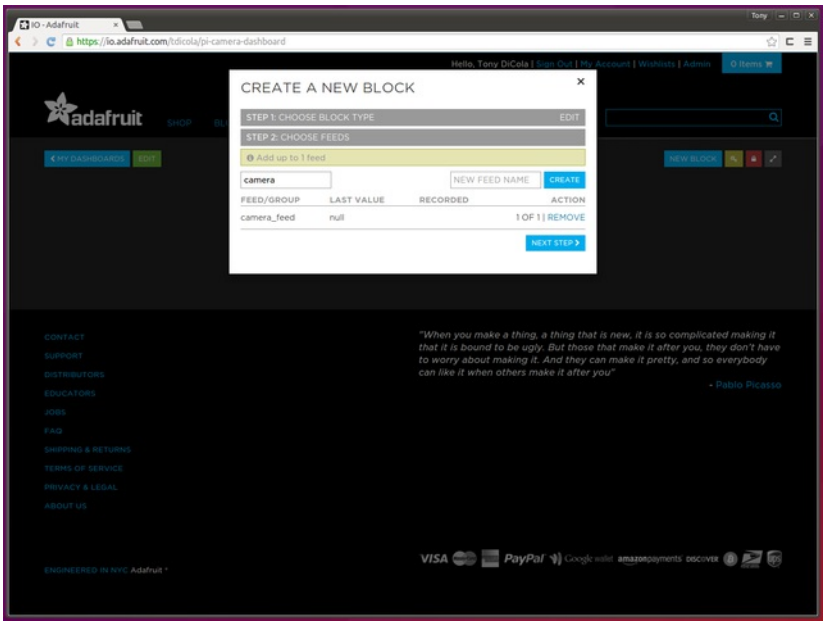
Once the dashboard is created click the **New Block '+'** button in the upper right to bring up the block selection list. Scroll down to the **image block** highlighted below and click **Create**:



In the block creation wizard that appears find the feed named**camera_feed** by scrolling down to it or using the filter/search box in the upper left. Then click the **Choose** action to select the feed.

Don't worry if you see the last value is null, you can ignore the value for now and check how the feed looks on the dashboard.

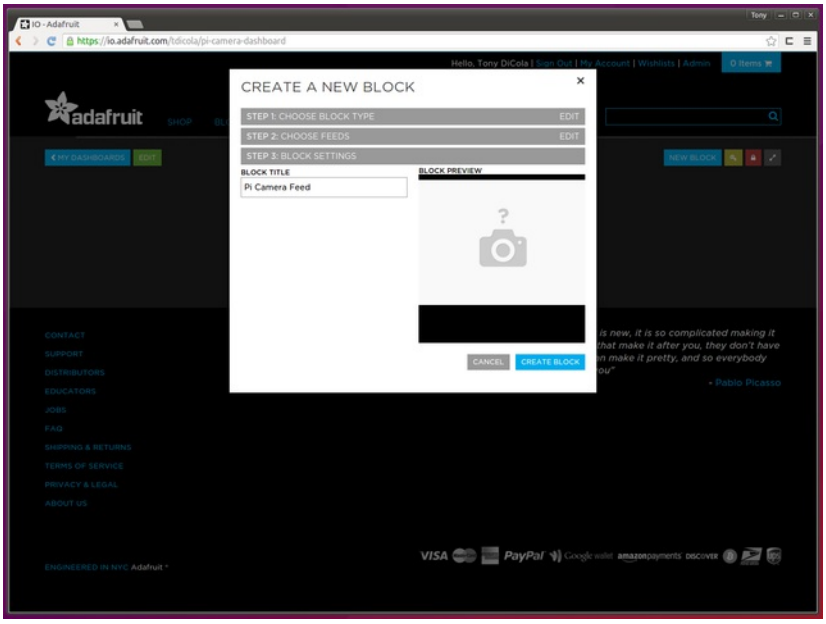
Once you've chosen the feed click**Next Step** to continue:



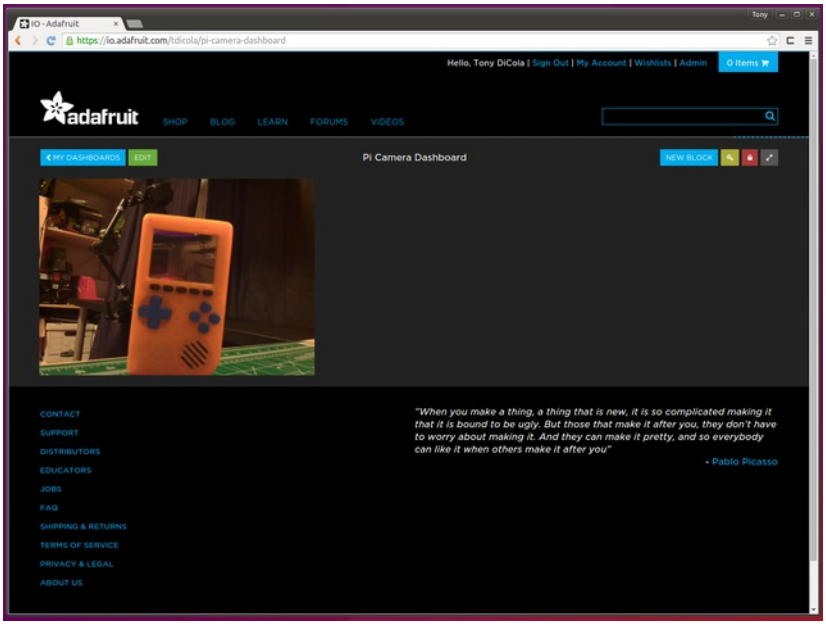
On the next page you can give the block a better title.

Don't worry if the block preview shows a blank image, you can continue moving forward.

Click **Create Block** to finish creating the image block:



You should see the block added to the dashboard and it start to display camera images:



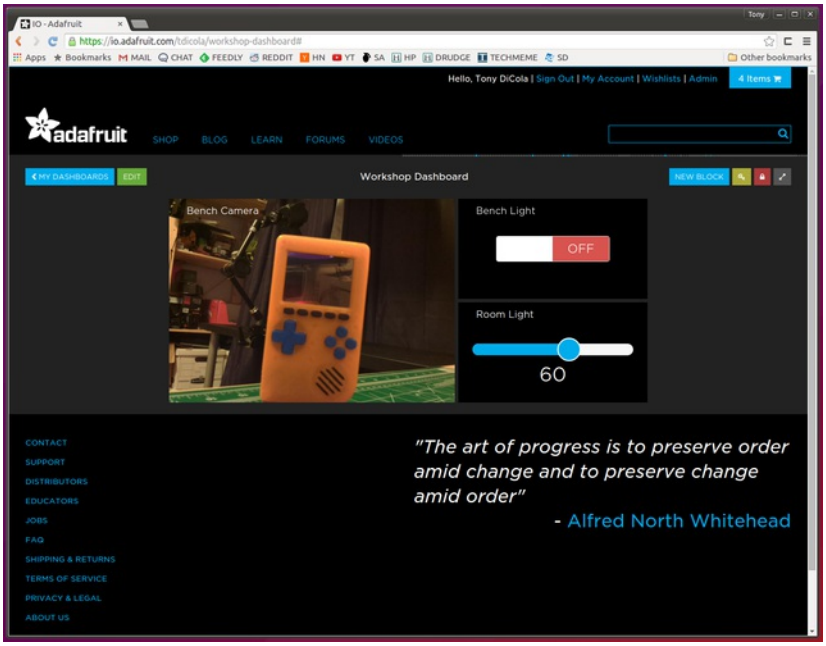
You might notice the image animating through different frames from the camera. The dashboard webpage is actually caching the most recently sent images and animating the display of them. This is handy to get a quick idea of what has recently happened in the feed. Note that this animation will only run from data stored in the browser cache--there's no actual history of image data stored in the feed, only the most recent image.

Awesome, now you have an internet dashboard with a live image feed! You can add more blocks with other camera feeds, sensor values, buttons, and more to build the perfect dashboard for your devices!

For example check out guides on using Adafruit IO to monitor and control devices like:

- Adafruit IO Basics: Digital Output (http://adafru.it/iRe)
- Using IFTTT with Adafruit IO to Make an IoT Door Detector(http://adafru.it/jdq)

You could build a dashboard to monitor and control the devices in your room:



Stop adafruit-io-camera

To stop the camera tool from sending data to Adafruit IO go back to a terminal on the Pi and run the following command:

```
adafruit-io camera stop
```

You should see the tool confirm that it stopped the camera:

```
[info] stopping camera...
```

Run adafruit-io-camera at Boot

To setup the camera tool to run automatically at boot you can call it from the /etc/rc.local file as a simple means of starting the tool.

Run the following command to edit this file:

```
sudo nano /etc/rc.local
```

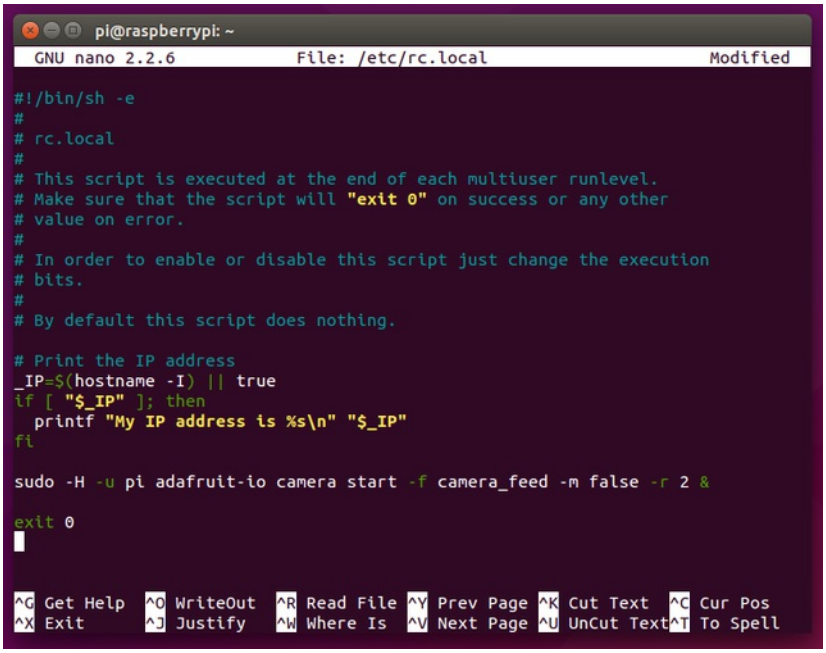
Now add a new line right **above** the exit 0 line at the end:

```
sudo -H -u pi adafruit-io camera start -f camera_feed -m false -r 2 &
```

This line will use the sudo command to switch to the default pi user (which has the Adafruit IO credentials associated with its home directory) and then run the camera start command (notice all the same parameters to control the tool are passed as before).

Be sure the lines ends with an ampersand '&' character! If you forget to add this character the Pi might not boot up!

Here's what the /etc/rc.local file should look like:



Now save the file and exit the editor by pressing**Ctrl-o** then **enter** and then **Ctrl-x**.

Reboot the Pi by running:

```
sudo reboot
```

When the Pi reboots it should start the camera tool and send data to Adafruit IO.

To disable the camera tool edit the /etc/rc.local file and remove the line that was added. Then save and reboot the Pi.

That's all there is to using the adafruit-io-camera tool to send Pi camera images to Adafruit IO!