RaspberryPi+Github Actions+CI/CD

Pabluc · Follow

6 min read · Apr 12, 2021

28

A small practice to spend my free time

Introduction

I was alone in my home and for a moment a thought goes through my mind.
I know we can use SSH or VNC to manage a Raspberry but I wondered if we can implement a process like CI/CD to update a program running in a device.

Image that we have a little webserver through a RaspberryPi. It could be inefficient to do the following things:

- Rewrite the code and push it to our repository. After that connect us to Raspberry, update the code, and restart the webserver.
- Keep synchronized the code of Raspberry and the repository if we don't have an automatic way to update our systems.
- Add the SSH Public Keys of all developers in the device.

So, What did I do?

I got my RaspberryPi where I used to do experiments. So, what do I needed to start?

1. Create a Github repository where we can upload an example of code. For example https://github.com/pabluc/test-gh-raspberry

2. Write the example where we can later change its behavior. I wanted to simulate a new deployment, so I used two leds with different colors so we can see the change. Here we can see how we can turn on a led.

The example is composed of two files. One of those is *Led.py* and the second is *Job.py*.

Led.py: It is responsible for flashing a led.

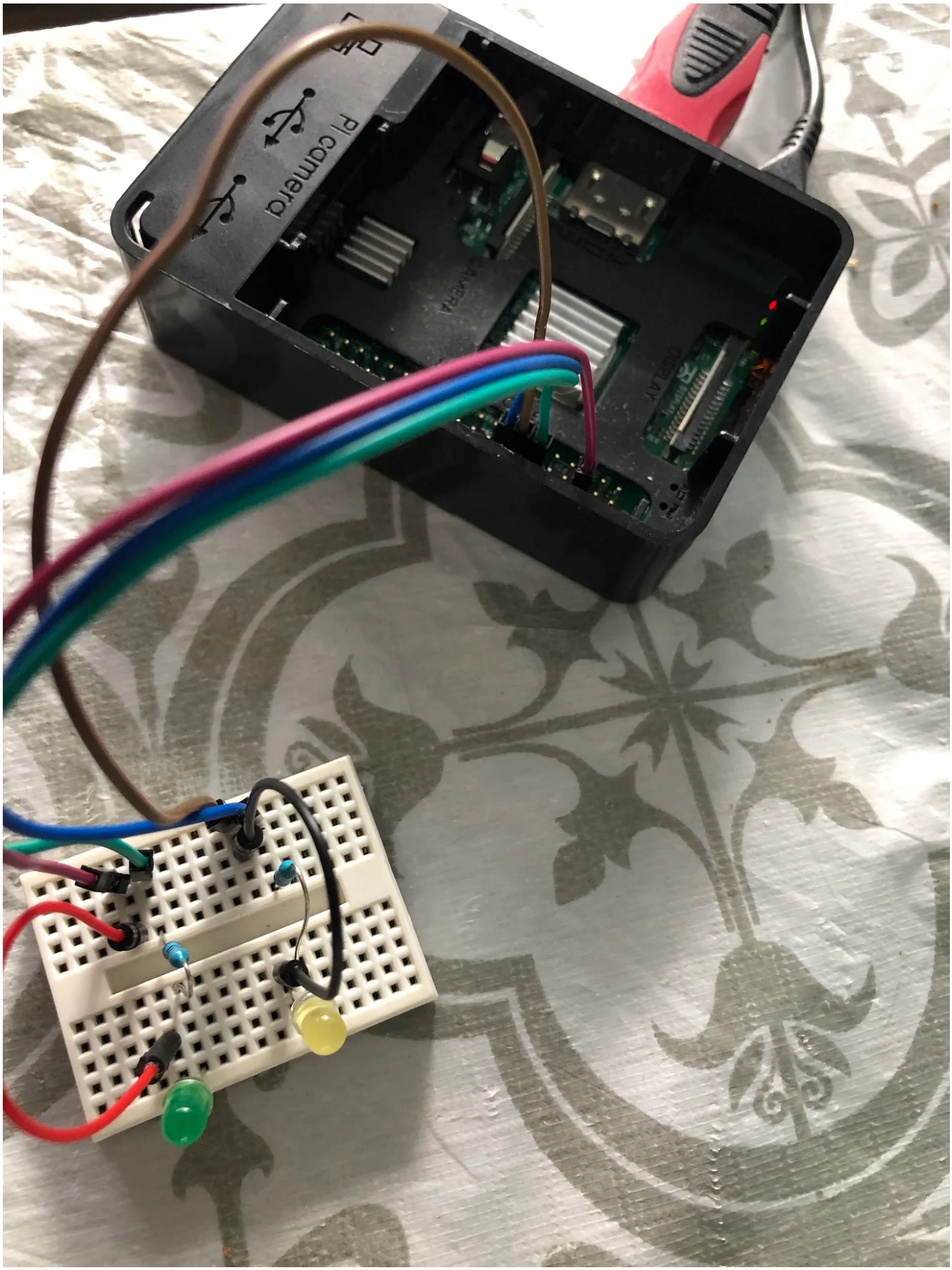
```
#Led.py

import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

pinout = 23
color = "Yellow"

GPIO.setup(pinout,GPIO.OUT)
print "LED on N" + str(pinout) + " " + color
GPIO.output(pinout,GPIO.HIGH)
time.sleep(1)
print "LED off N" + str(pinout) + " " + color
GPIO.output(pinout,GPIO.LOW)
time.sleep(1)
```

Job.py: It is responsible for running *Led.py*

```
#Job.py
```

```
while True:
  execfile('led.py')
```

Before of continue I recommend test the files separately. We need to be sure that the led can be turn on correctly. For example, We can run *Led.py* and see the led blinking.



RaspberryPI and Leds

3. Prepare our repository and device to run the Github Actions in a self-hosted way. From our repository: Settings / Actions, we add a self-hosted

runner.

but What is a Github Action?

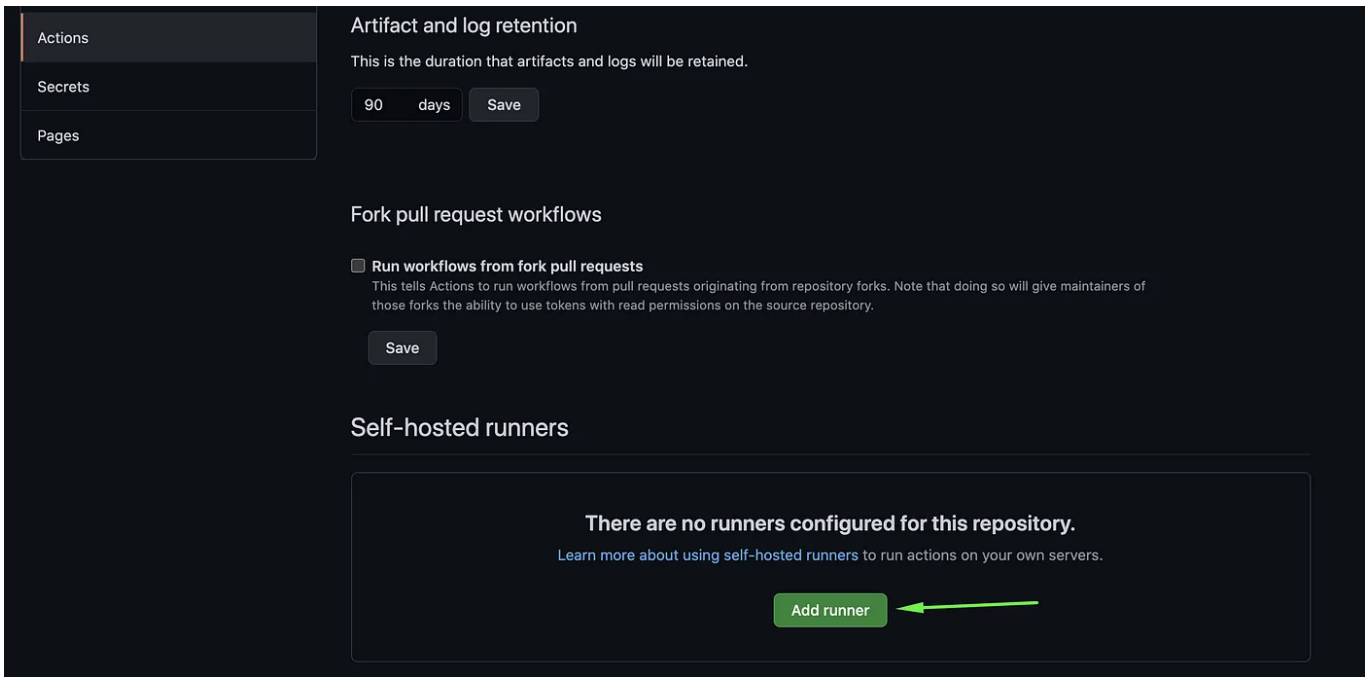
> *“GitHub Actions*
>
> *Automate, customize, and execute your software development workflows right in your repository with GitHub Actions. You can discover, create, and share actions to perform any job you’d like, including CI/CD, and combine actions in a completely customized workflow.” https://docs.github.com/en/actions*

The concept is very broad, so we can summarize it: they are instructions that we can run event when an event is triggered. It could be a single commit, a push, a new pull request, or others. The instructions have the power to analyze our code and do many things.

and…What is a self-hosted runner?

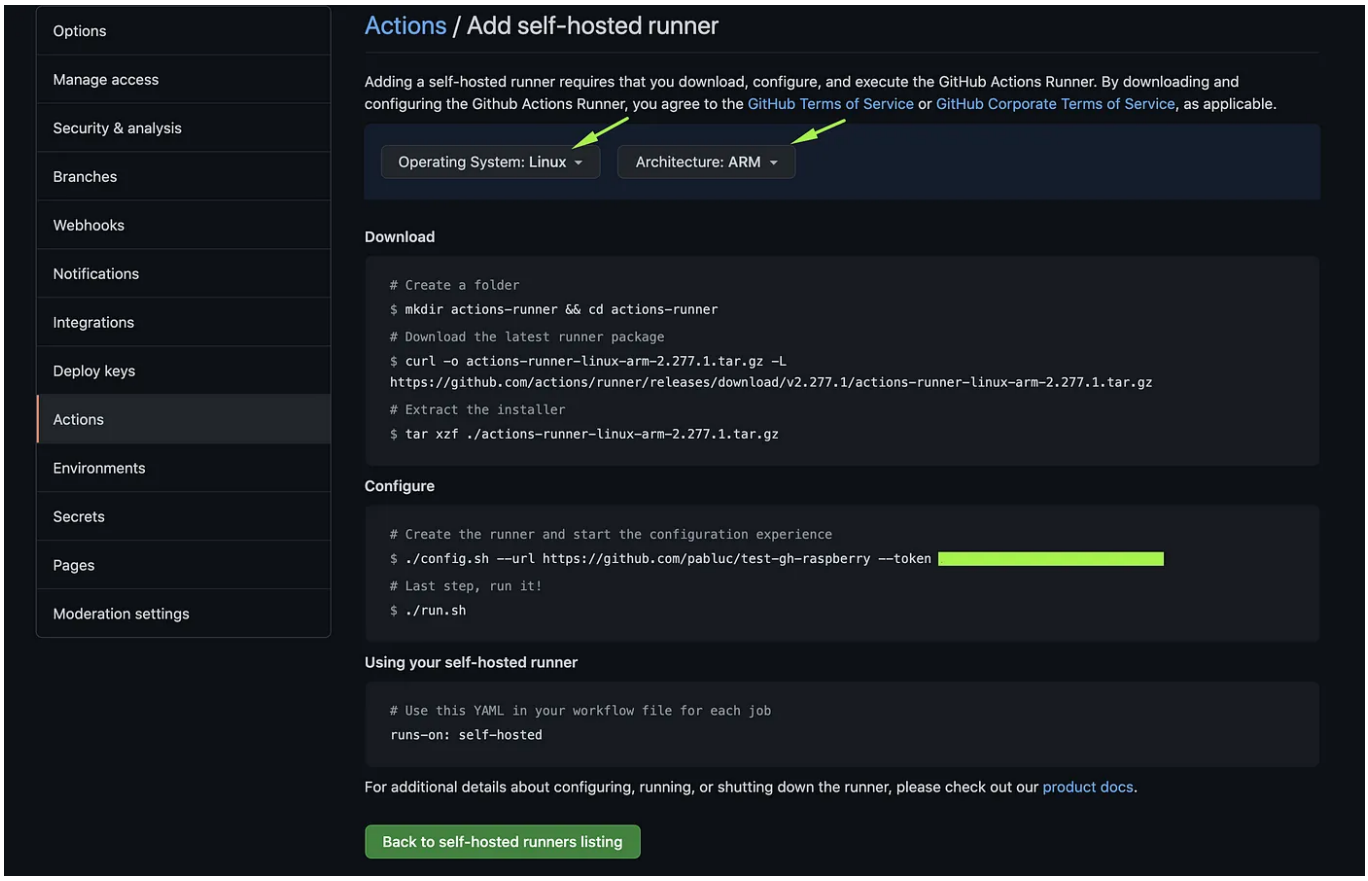
> *“About self-hosted runners*
>
> *Self-hosted runners offer more control of hardware, operating system, and software tools than GitHub-hosted runners provide. With self-hosted runners, you can choose to create a custom hardware configuration with more processing power or memory to run larger jobs, install software available on your local network, and choose an operating system not offered by GitHub-hosted runners. Self-hosted runners can be physical, virtual, in a container, on-premises, or in a cloud….” https://docs.github.com/en/actions/hosting-your-own-runners/about-self-hosted-runners*

Basically one of the places where we can run the GitHub actions is from our hardware. We can check out the code and embedded in our operating system.



Add self-hosted runner

In the following step, we need to choose Linux and ARM because in this example we are using a RaspberryPI in 32 bits.



Get the commands to setup our RaspberryPi

We need to run the commands in our Raspberry. For this, we can be on the device or connect through ssh.

After we run the commands, we can see the following:

```
pi@raspberrypi:~/actions-runner $ ./config.sh --url https://github.com/[redacted]
spberry --token [redacted]

--------------------------------------------------------------------------------
|        ____ _ _   _   _       _          _        _   _                      |
|       / ___(_) |_| | | |_   _| |__      / \   ___| |_(_) ___  _ __  ___      |
|      | |  _| | __| |_| | | | | '_ \    / _ \ / __| __| |/ _ \| '_ \/ __|     |
|      | |_| | | |_|  _  | |_| | |_) |  / ___ \ (__| |_| | (_) | | | \__ \     |
|       \____|_|\__|_| |_|\__,_|_.__/  /_/   \_\___|\__|_|\___/|_| |_|___/     |
|                                                                              |
|                       Self-hosted runner registration                        |
|                                                                              |
--------------------------------------------------------------------------------

# Authentication


√ Connected to GitHub

# Runner Registration

Enter the name of runner: [press Enter for raspberrypi]

This runner will have the following labels: 'self-hosted', 'Linux', 'ARM'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip]

√ Runner successfully added
√ Runner connection is good

# Runner settings

Enter name of work folder: [press Enter for _work]

√ Settings Saved.

pi@raspberrypi:~/actions-runner $
```

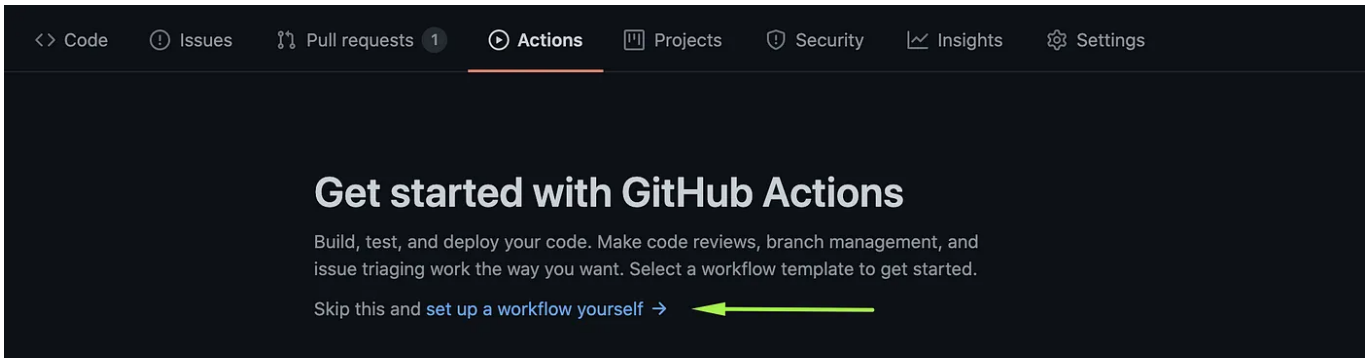
Self-hosted runner installed.

Keep present the working folder. For this example we will leave the default value, it will be used later.

```
pi@raspberrypi:~/actions-runner $ ./run.sh

√ Connected to GitHub

2021-04-10 20:55:18Z: Listening for Jobs
```

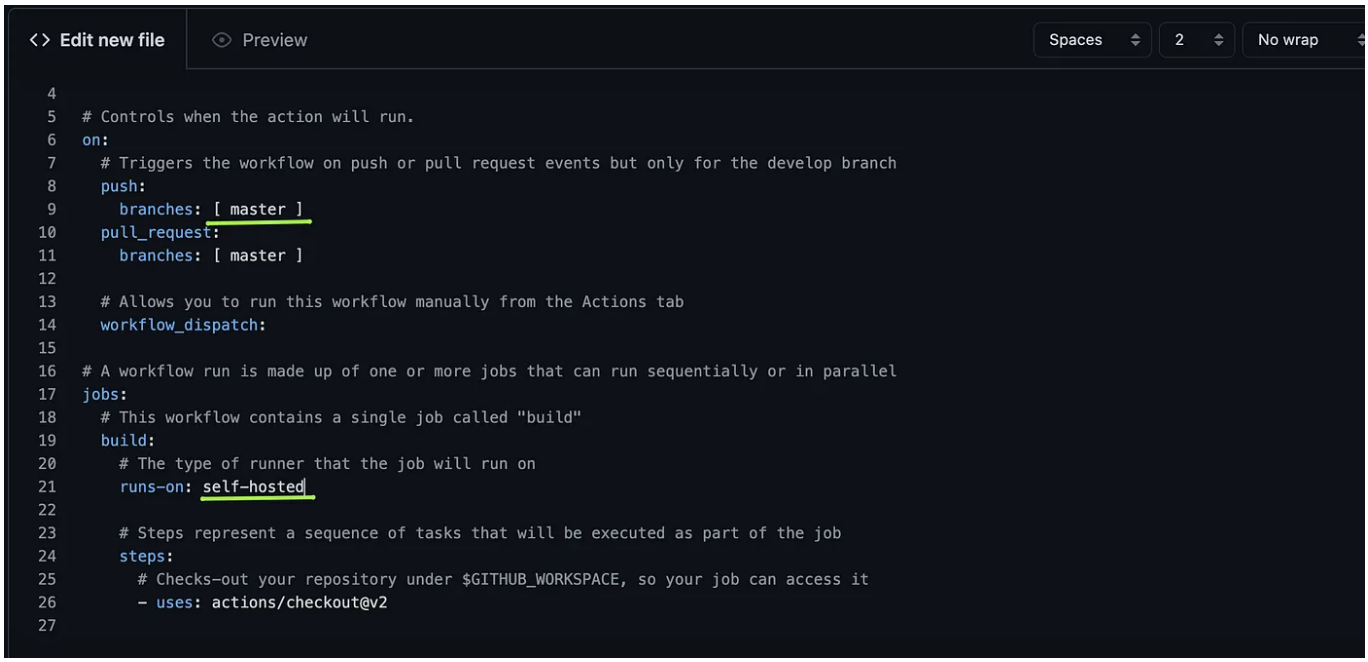
Listening for Jobs

4. We need to add a GitHub action to trigger the deployment. For this, we go back to the repository, go to Actions and set up a workflow.



Go to create the GitHub action

In this case, we will use the master branch but we can use any other branch. Also, we need to setup as self-hosted so the action can run on our device.



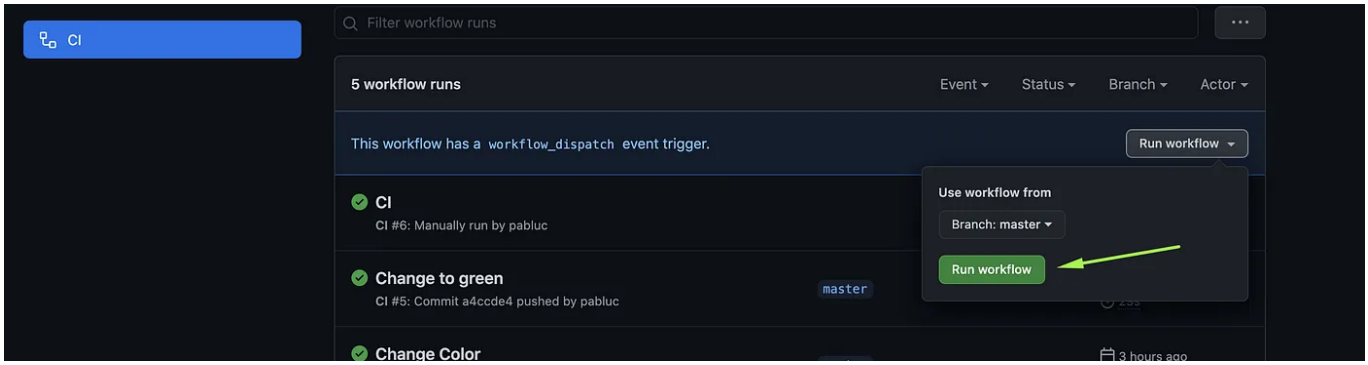
Change to use “master” branch and “self-hosted”.

After we committed our GitHub action we can do `git pull origin master` and see how our GitHub Action is embedded in our repository:

```
→  test-gh-raspberry git:(master) ls .github/workflows
main.yml
```

GitHub Action

5. Now, we will force the GitHub action so we can see it's working and see how the Raspberry starts the first job.



Force the github action

```
pi@raspberrypi:~/actions-runner $ ./run.sh

√ Connected to GitHub

2021-04-10 21:46:59Z: Listening for Jobs
2021-04-10 21:48:37Z: Running job: build
```

Running our Github action

6. After the previous step, now we can see the code in the following folder

```
pi@raspberrypi:~/actions-runner/_work/test-gh-raspberry/test-gh-raspberry $ ls
job.py  led.py
pi@raspberrypi:~/actions-runner/_work/test-gh-raspberry/test-gh-raspberry $
```

We need to found the files where they were downloaded. So we can run our job.py and run the solution.

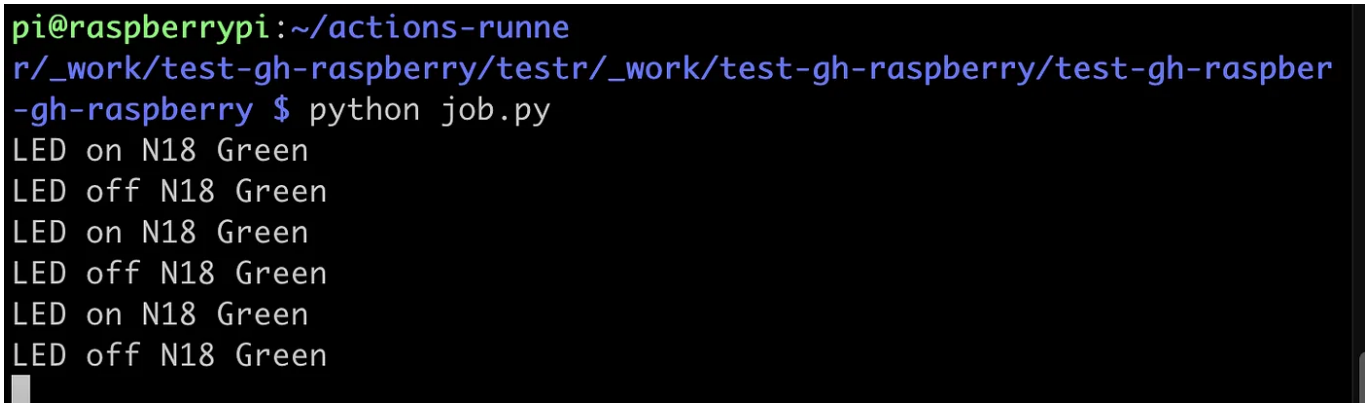
Remember the previous steps where we installed our self-hosted runner? It's here where we should found the folder of our code. The `_work` folder

has a clone of our repository. Is here where the self-hosted runner is downloading the code each time occur an event.

7. Run solution, standing where the GitHub action downloaded the code we can run the following command:

```
python job.py
```

then, we can see it working.



Running the solution

Remember, we have two files, if we want to change the led that we are blinking we should do a new commit updating *Led.py* and wait for the Github actions to do his work. Github will send a signal to the self-host runner on our Raspberry, The GitHub action has a sentence to run actions/checkout@v2, so our code will be updated in the work folder. Immediately *Job.py* will start processing the new code because it is executing *Led.py with the following:*

```
#Job.py

while True:
  execfile('led.py')
```

For this I leave a video where we can see the complete logic:

https://youtu.be/cqeuFC6ng28

Conclusions

- It was a great practice, there are many parts of this small example where we can deep. For example, Github's actions are amazing and each day I can see his power. I left in the bibliography some articles to learn concepts used in this post.
- We can apply different flavors to develop this practice. For example, we can use cronjob's or a process as a service to run the functionality. So we can replace our *Job.py* for software made it for that.
- Also, it's an interesting thing to think, what happens if something is wrong in our delivery. I mean, When we are deploying the application we should keep in mind that we could be using the system at the same moment. So, We could improve this part, For example: after download the code and checked his integrity we can restart the *Job.py*
- As my first post on medium, I found a good tool to continue writing examples and ideas.

Bibliography:

- https://thepihut.com/blogs/raspberry-pi-tutorials/27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins
- https://elinux.org/RPi_Low-level_peripherals

- https://www.tutorialspoint.com/How-can-I-make-one-Python-file-run-another
- https://dev.to/l04db4l4nc3r/self-hosted-github-actions-using-raspberry-pi-212m
- https://docs.github.com/en/actions/hosting-your-own-runners/about-self-hosted-runners