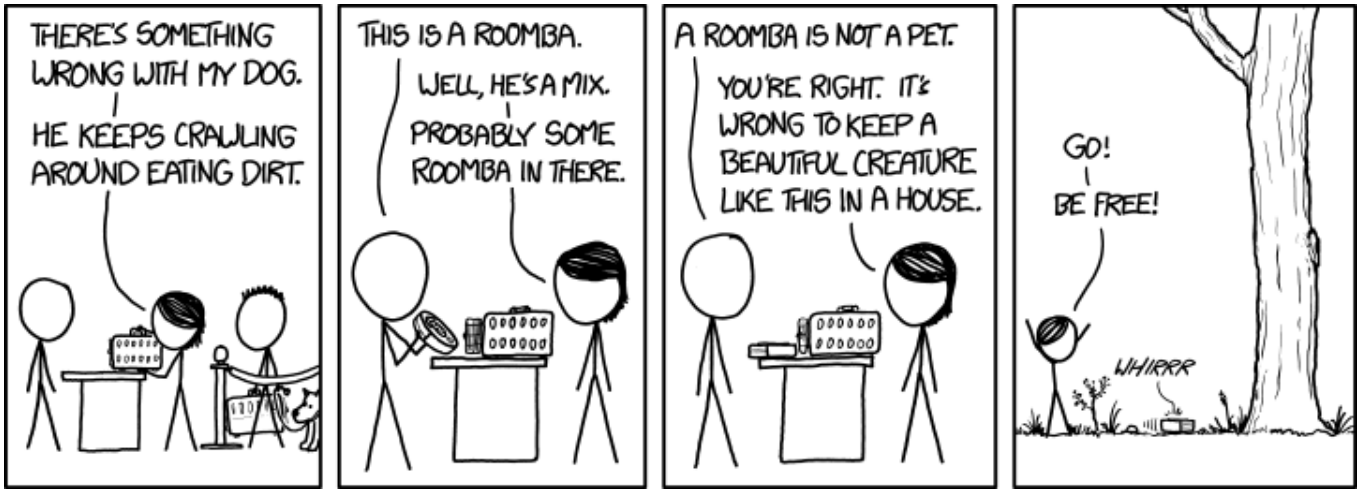
How to create a watchdog in Python to look for filesystem changes

2019-01-13

BY DAVIDE MASTROMATTEO

|5 minute read|Dev|#check , #Filesystem , #Python , #watchdog



Hey guys, today's post is about how to create a watchdog in Python. But what is a "watchdog"?

A watchdog is a little piece of software that monitors our filesystem looking for any changes (like the creation, change or deletion of a file or of a directory). When a change occurs, the watchdog report it to us raising a specific event that we can handle.

For example, let's suppose you have developed a program that use a configuration file. Your program could set a watchdog to monitor that file and if the configuration file is modified you could think to reload it and apply the new configuration at runtime, without the need of restarting your program.

That's cool, uh?

So, today we will code a watchdog in Python.

To code this program we will need an additional module called "watchdog" (wow, who could have guessed it?) written by Yesudeep Mangalapilly, so let's start by installing it. As always, I raccomand to use virtual environments instead of installing packages system wide. If you want to find out more about virtual environments (that's probabilly because you haven't read all my previous post, so shame on you!), just have a look at this article.

```
pip install watchdog
```

Done, we have almost finished, haven't we?
I mean, we still need to code the program that will actually use this module, but trust me, that will be really easy!

Ok, let's start and pretend that we want to create a program that logs all the file that are created or modified in the same directory of our program.

Step 1. Import some stuff

```
1 import time
2 from watchdog.observers import Observer
3 from watchdog.events import PatternMatchingEventHandler
```

I won't explain these imports right now, they will be clear in a moment.

Step 2. Create the event handler

The event handler is the object that will be notified when something happen on the filesystem you are monitoring.

```
1 if __name__ == "__main__":
2     patterns = ["*"]
3     ignore_patterns = None
4     ignore_directories = False
5     case_sensitive = True
6     my_event_handler = PatternMatchingEventHandler(patterns, ignore_patterns, ig
```

In my example I have used some variables just to made the configuration of the event handler a little bit easier to be undestood.
The "patterns" variable contains the file patterns we want to handle (in my sce-

that we can set to True if we want to be notified just for regular files (not for directories) and the "case_sensitive" variable is just another boolean that, if set to "True", made the patterns we previously introduced "case sensitive" (that's normally a good idea, unless you are working with stupid case-insensitive-file-systems… yeah, I'm talking about you Windows! :P).

Step 3. Handle all the events

Now that we have created the handler we need to write the code we want to run when the events are raised.
So, let's start creating four different functions that will be used when a file is modified, created, deleted or moved.

copy

```
def on_created(event):
    print(f"hey, {event.src_path} has been created!")

def on_deleted(event):
    print(f"what the f**k! Someone deleted {event.src_path}!")

def on_modified(event):
    print(f"hey buddy, {event.src_path} has been modified")

def on_moved(event):
    print(f"ok ok ok, someone moved {event.src_path} to {event.dest_path}")
```

In our example we will just print out something regarding the events that has been captured.
It goes without saying that this is just an example, we could execute any kind of code instead of these useless print functions…

And now, under the creation of our event handler (the code of the previous step) we need to specify to the handler that we want these functions to be called when the corresponding event is raised

copy

```
my_event_handler.on_created = on_created
    my_event_handler.on_deleted = on_deleted
```

Step 4. Create an observer

We need another object, known as the observer, that will monitor our filesystem, looking for changes that will be handled by the event handler.

Let’s create it right now.

copy

```
path = "."
go_recursively = True
my_observer = Observer()
my_observer.schedule(my_event_handler, path, recursive=go_recursively)
```

As you can see, I have just created an object of the Observer class and then called the schedule method, passing to it:

- the event handler that will handle the event
- the path to be monitored (in my case is “.”, that’s the current directory)
- a boolean that allow me to catch all the event that occurs even in the sub directories of my current directory.

Step 5. Start the observer

Ok, we almost finished our program, now we need just to start the observer thread.

copy

```
my_observer.start()
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    my_observer.stop()
    my_observer.join()
```

I think this snippet doesn’t need any comments, does it? We are just starting the observer thread to get all the events.

Testing our new shiny watchdog

And now it's showtime guys, let's test our watchdog!

Start the program from a terminal and you will see your cursor just hang. That's a good start.

Now open a different terminal, move under the directory of your program and try to do something. For example, I have started vim (my favourite editor) to create a file:

```
$ vim dave.txt
```

by the time vim started, in my running console some statement has been printed:

```
hey, ./.dave.txt.swp has been created!
hey buddy, . has been modified
```

Wow, it works! When you try to create a file with Vim it start by creating a .swp file and that's what our program has detected, printing the first line. But why the second line? Well, if the current directory has a new file we can say that it's changed as well, can't we?

Now I will try to save my file and…

```
hey, ./dave.txt has been created!
hey buddy, . has been modified
```

Ok, Vim has created my file (dave.txt) and my program has detected it! Well done!

Now I quit vim (for the record and for all the "Windows guys" that are currently reading this… you don't need to unplug the power cord of your pc to quit vim, it's

```
what the f**k! Someone deleted ./.dave.txt.swp!
hey buddy, . has been modified
```

Ok, vim has deleted the temporary .swp file and the program has detected it. Mission accomplished! :)

The bottom line

In this article you have seen how easy it could be to set up a watchdog in Python that monitors a filesystem and react when a change occurs.

If you want to find out more about the watchdog module, you can have a look at the official documentation.

Happy Coding! D.

Did you find this article helpful?