How to make a Python script run like a service or daemon in Linux

Ask Question

154

80

I have written a Python script that checks a certain e-mail address and passes new e-mails to an external program. How can I get this script to execute 24/7, such as turning it into daemon or service in Linux. Would I also need a loop that never ends in the program, or can it be done by just having the code re executed multiple times?

python linux scripting daemons

edited Apr 27 '16 at 14:04
Martin Thoma
45.2k 62 324 546

asked Oct 21 '09 at 19:36
adhanlon
2,684 11 35 40

1 See SO question: stackoverflow.co

3 "checks a certain e-mail address and passes new e-mails to an external program" Isn't that what sendmail does? You can define mail alias to route a mailbox to a script. Why aren't you using mail aliases to do this? – S.Lott Oct 21 '09 at 19:54

1 On a modern linux which has `systemd` you can create a systemd service in `daemon` mode as described here. See also: freedesktop.org/software/systemd/man/systemd.service.html – ccpizza Sep 11 '18 at 23:22

If the linux system supports systemd, use the approach outlined here. – gerardw Oct 31 '18 at 18:42

13 Answers

89 ✓

You have two options here.

1. Make a proper **cron job** that calls your script. Cron is a common name for a GNU/Linux daemon that periodically

set. You add your script into a crontab or place a symlink to it into a special directory and the daemon handles the job of launching it in the background. You can read more at wikipedia. There is a variety of different cron daemons, but your GNU/Linux system should have it already installed.

2. Use some kind of **python approach** (a library, for example) for your script to be able to daemonize itself. Yes, it will require a simple event loop (where your events are timer triggering, possibly, provided by sleep function).

I wouldn't recommend you to choose 2., because you're in fact repeating cron multiple simple tools interact and solve your problems. Unless there are additional reasons why you should make a daemon (in addition to trigger periodically), choose the other approach.

Also, if you use daemonize with a loop and a crash happens, noone will check the mail after that (as pointed out by [Ivan Nevostruev]() in comments to [this]() answer). While if the script is added as a cron job, it will just trigger again.

dited May 23 '17 at 12:02
Community ♦
1 1

nswered Oct 21 '09 at 19:43
P Shved
73.2k 12 107 154

6 +1 to the cronjob. I don't think the question specifies that it is checking a local mail account, so mail filters do not apply – John La Rooy Oct 21 '09 at 21:10

What happen does use a loop without termination in a set up such `.py` for hourly, will it create many processes that will never be terminated? If so, I think this would quite like daemon. – Veck Hsiao Jan 14 '16 at 8:47

I can see that cron is an obvious solution if you check check for emails once a minute (which is the lowest time resolution for Cron). But what if I want to check for emails every 10 seconds? Should I write the Python script to run query 60 times, which means it ends after 50 seconds, and then let cron start the script again 10 seconds later? – Mads Skjern Mar 10 '16 at 10:29

I have not worked with daemons/services, but I was under the impression that it (OS/init/init.d/upstart or what it is called) takes care of restarting a daemon when/if it ends/crashes. – Mads Skjern Mar 10 '16 at 10:32

@VeckHsiao yes, crontab calls a script so

loop.... – Pipo
May 24 '18 at
14:47

67

Here's a nice class
that is taken from
here:

```
#!/usr/bin/env py

import sys, os, t
from signal impor

class Daemon:
        """
        A generic

        Usage: su
        """
        def __ini
stderr='/dev/null
                s
                s
                s
                s

        def daemo
                "
                d
                P
0201563177)
                h
                "
                t




                e

e.strerror))


                #
                o
                o
                o

                #
                t
                #
                s
                s
                s
                s
                s
                o
                o
                o

                #
                a
                p
                f

        def delpi
                o

        def start
                "
                S
                "
                #
                t



                e



                i

running?\n"



                #
                s
                s

        def stop(
                "
                S
                "
                #
                t



                e



                i

running?\n"



                #
                t
        def resta
                "
                R
                "
                s
                s

        def run(s
                "
                Y
will be called af
                d
                "
```

dited Apr 19 '16 at 5:00

Dain42
5 2

nswered Jun 16 '11 at 15:58

the_drow
8,884 19 102 176

50

You should use the python-daemon library, it takes care of everything.

From PyPI: *Library to implement a well-behaved Unix daemon process.*

dited Aug 30 '16 at 15:53

gonz
3,630 2 31 50

nswered Oct 21 '09 at 19:43

Prody
3,429 6 37 61

2 Ditto Jorge Vargas's comment. After looking at the code, it actually looks like quite a nice piece of code, but the

use, which means most developers will rightfully ignore it for better documented alternatives. – Cerin Mar 16 '12 at 14:52

20 The docs can be found here: python.org/dev/peps/pep-3143 – Alan Hamlett Jun 5 '13 at 7:36

Seems not to work properly in Python 3.5: gist.github.com/MartinThoma/fa4deb2b4c71ffcd726b24b7ab581ae2 – Martin Thoma Dec 7 '17 at 7:49

37

You can use fork() to detach your script from the tty and have it continue to run, like so:

```
import os, sys
fpid = os.fork()
if fpid!=0:
  # Running as da
  sys.exit(0)
```

Of course you also need to implement an endless loop, like

```
while 1:
  do_your_check()
  sleep(5)
```

Hope this get's you started.

ıswered Oct 21 '09 at 19:45

Hello, I've tried this and it works for me. But when I close the terminal or get out of the ssh session, the script also stops working!! – David Okwii Nov 21 '16 at 12:15

@DavidOkwii `nohup` / `disown` commands would detach process from console and it won't die. Or you could start it with init.d – pholat Sep 7 '17 at 12:32

12

You can also make the python script run as a service using a shell script. First create a shell script to run the python script like this (scriptname arbitary name)

```
#!/bin/sh
script='/home/..
/usr/bin/python $
```

now make a file in /etc/init.d/scriptname

```
#! /bin/sh

PATH=/bin:/usr/bi
DAEMON=/home/.. p
PIDFILE=/var/run/

test -x $DAEMON |

. /lib/lsb/init-f

case "$1" in
  start)
     log_daemon_m
```

```
        killproc -p
        PID=`ps x |g
        kill -9 $PID
        log_end_msg
      ;;
     force-reload|re
        $0 stop
        $0 start
      ;;
     status)
        status_of_pr
      ;;
   *)
      echo "Usage: /
      exit 1
     ;;
  esac

  exit 0
```

Now you can start and stop your python script using the command /etc/init.d/scriptname start or stop.

ıswered Oct 22 '13 at 9:56

Kishore K
900 2 8 16

I just tried this, and it turns out this will start the process, but it will not be daemonized (i.e. it's still attached to the terminal). It would probably work fine if you ran update-rc.d and made it run on boot (I assume there's no terminal attached when these scripts are run), but it doesn't work if you invoke it manually. Seems like supervisord might be a better solution. – ryuusenshi May 23 '14 at 0:02

9

many purposes. However it doesn't create a service or daemon as you requested in the OP. `cron` just runs jobs periodically (meaning the job starts and stops), and no more often than once / minute. There are issues with `cron` -- for example, if a prior instance of your script is still running the next time the `cron` schedule comes around and launches a new instance, is that OK? `cron` doesn't handle dependencies; it just tries to start a job when the schedule says to.

If you find a situation where you truly need a daemon (a process that never stops running), take a look at `supervisord` . It provides a simple way to wrapper a normal, non-daemonized script or program and make it operate like a daemon. This is a much better way than creating a native Python daemon.

ıswered Oct 22 '13 at 10:36

Chris Johnson
12.8k 3 54 60

9

supported version is Deamonize Install it from Python Package Index (PyPI):

```
$ pip install dae
```

and then use like:

```
...
import os, sys
from daemonize im
...
def main()
      # your code

if __name__ == '_
        myname=os
        pidfile='
        daemon = 
        daemon.st
```

dited May 4 '17 at 17:30

Gal Bracha
7,695 5 46 63

nswered Apr 3 '16 at 11:08

fcm
576 8 19

8

how about using `$nohup` command on linux?

I use it for running my commands on my Bluehost server.

Please advice if I am wrong.

dited Jan 21 '12 at 21:07

Udo Held
9,417 11 51 79

nswered Jan 21 '12 at 21:00

faisal00813
309 4 9

3

alias will do this inside the mail system without you having to fool around with daemons or services or anything of the sort.

You can write a simple script that will be executed by sendmail each time a mail message is sent to a specific mailbox.

See http://www.feep.net/sendmail/tutorial/intro/aliases.html

If you really want to write a needlessly complex server, you can do this.

```
nohup python mysc
```

That's all it takes. Your script simply loops and sleeps.

```
import time
def do_the_work()
    # one round o
while True:
    time.sleep( 6
    try:
        do_the_wo
    except:
        pass
```

lited Oct 21 '09 at 20:11

ıswered Oct 21 '09 at 19:44

S.Lott
322k 69 443 720

6 The problem

run it again – Ivan Nevostruev Oct 21 '09 at 19:51

if the function do_the_work() crashes, it would be called again after 10 minutes, since only the one function call raises an error. But instead of crashing the loop just the `try` part fails and the `except:` part will be called instead (in this case nothing) but the loop will continue and keep trying to call the function. – sarbot May 6 '18 at 17:07

3

If you are using terminal(ssh or something) and you want to keep a long-time script working after you log out from the terminal, you can try this:

`screen`

`apt-get install screen`

create a virtual terminal inside(namely abc): `screen -dmS abc`

now we connect to abc: `screen -r abc`

So, now we can run python script: `python Keep_sending_mail` from now on, you can directly close your terminal, however, the python script will keep running rather than being shut down

> Since this `Keep_sending_mail.py` 's PID belong to the virtual screen rather than the terminal(ssh)

If you want to go back check your script running status, you can use `screen -r abc` again

ıswered Jan 26 '16 at 6:59

Microos
848 2 12 27

1 while this works, it is very quick and dirty and should be avoided in production – pcnate Aug 16 '17 at 0:08

1

Use whatever service manager your system offers - for example under Ubuntu use **upstart**. This will handle all the details for you such as start on boot, restart on crash, etc.

1

I would recommend this solution. You need to inherit and override method `run`.

```
import sys
import os
from signal import impor
from abc import A


class Daemon(obje
    __metaclass__


    def __init__(
        self._pid


    @abstractmeth
    def run(self)
        pass


    def _daemoniz
        # decoupl
        pid = os.

        # stop fi
        if pid > 
            sys.e

        # write p
        with open
            print


    def start(sel
        # if daem
        if os.pat
            raise

        # create 
        self._dae

        # run the
        self.run(


    def stop(self
        # check t
        if os.pat
            # rea
            with 
                p

            # rem
            os.re
```

```
                raise


        def restart(s
            self.stop
            self.star
```

dited May 9 '15 at 21:31

nswered May 8 '15 at 11:12

Fomalhaut
2,581 2 14 30

0

to creating some thing that is running like service you can use this thing :

The first thing that you must do is installing the Cement framework: Cement frame work is a CLI frame work that you can deploy your application on it.

command line interface of the app :

interface.py

```
    from cement.core
    from cement.core
    from YourApp imp

    class Meta:
        label = 'base
        description =
        arguments = [
            (['-r' ,
              dict(ac
            (['-v', '
              dict(ac
            ]
            (['-s', '
              dict(ac
            ]

        @expose(hide=
            #Stop
            YourA

class App(Cement
      class Meta
      label = 'U
      base_contr
      handlers =

with App() as ap
      app.run()
```

YourApp.py class:

```
import threading

class yourApp:
    def __init__
       self.loge
       thread = 
       thread.da
       thread.st

    def start(se
       #Do every
       pass
    def stop(sel
       #Do some 
```

Keep in mind that your app must run on a thread to be daemon

To run the app just do this in command line

> python interface.py --help

dited Aug 7 '17 at 10:00

nswered Aug 7 '17 at 9:52

Manouchehr Rasouli
83 7