Why is my crontab not working, and how can I troubleshoot it?

Asked 10 years, 1 month ago Modified 6 months ago Viewed 840k times

369

Want to improve this post? Provide detailed answers to this question, including citations and an explanation of why your answer is correct. Answers without enough detail may be edited or deleted.

This is a Canonical Question about using cron & crontab.

You have been directed here because the community is fairly sure that the answer to your question can be found below. If your question is not answered below then the answers will help you gather information that will help the community help you. This information should be edited into your original question.

The answer for '*Why is my crontab not working, and how can I troubleshoot it?*' can be seen below. This addresses the `cron` system with the crontab highlighted.

linux cron

4.6k watchers 37.8k questions

Linux is the generic term for a UNIX-like open source operating system based on the Linux kernel. View tag

4 This is a huge dupe of Reasons why crontab does not work on AskUbuntu. – Dan Dascalescu Apr 26, 2017 at 7:03

7 Answers

Sorted by: Highest score (default)

How to fix all of your crontab related woes/problems (Linux)

492

This is a community wiki, if you notice anything incorrect with this answer or have additional information then please edit it.

First, basic terminology:

- cron(8) is the daemon that executes scheduled commands.
- crontab(1) is the program used to modify user crontab(5) files.

- crontab(5) is a per user file that contains instructions for cron(8).

Next, education about cron:

Every user on a system may have their own crontab file. The location of the root and user crontab files are system dependant but they are generally below `/var/spool/cron` .

There is a system-wide `/etc/crontab` file, the `/etc/cron.d` directory may contain crontab fragments which are also read and actioned by cron. Some Linux distributions (eg, Red Hat) also have `/etc/cron.{hourly,daily,weekly,monthly}` which are directories, scripts inside which will be executed every hour/day/week/month, with root privilege.

root can always use the crontab command; regular users may or may not be granted access. When you edit the crontab file with the command `crontab -e` and save it, crond checks it for basic validity but does not guarantee your crontab file is correctly formed. There is a file called `cron.deny` which will specify which users cannot use cron. The `cron.deny` file location is system dependent and can be deleted which will allow all users to use cron.

If the computer is not powered on or crond daemon is not running, and the date/time for a command to run has passed, crond will not catchup and run past queries.

crontab particulars, how to formulate a command:

A crontab command is represented by a single line. You cannot use `\` to extend a command over multiple lines. The hash (`#`) sign represents a comment which means anything on that line is ignored by cron. Leading whitespace and blank lines are ignored.

Be VERY careful when using the percent (`%`) sign in your command. Unless they are escaped `\%` they are converted into newlines and everything after the first non-escaped `%` is passed to your command on stdin.

There are two formats for crontab files:

- User crontabs

```
# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7)
# |  |  |  |  |
# *  *  *  *  *   command to be executed
```

- System wide `/etc/crontab` and `/etc/cron.d` fragments

```
# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7)
# |  |  |  |  |
# *  *  *  *  * user-name  command to be executed
```

Notice that the latter requires a user-name. The command will be run as the named user.

The first 5 fields of the line represent the time(s) when the command should be run. You can use numbers or where applicable day/month names in the time specification.

- The fields are separated by spaces or tabs.
- A comma (`,`) is used to specify a list e.g 1,4,6,8 which means run at 1,4,6,8.
- Ranges are specified with a dash (`-`) and may be combined with lists e.g. 1-3,9-12 which means between 1 and 3 then between 9 and 12.
- The `/` character can be used to introduce a step e.g. 2/5 which means starting at 2 then every 5 (2,7,12,17,22...). They do not wrap past the end.
- An asterisk (`*`) in a field signifies the entire range for that field (e.g. `0-59` for the minute field).
- Ranges and steps can be combined e.g. `*/2` signifies starting at the minimum for the relevant field then every 2 e.g. 0 for minutes(0,2...58), 1 for months (1,3 ... 11) etc.

Debugging cron commands

Check the mail!

By default cron will mail any output from the command to the user it is running the command as. If there is no output there will be no mail. If you want cron to send mail to a different account then you can set the MAILTO environment variable in the crontab file e.g.

```
MAILTO=user@somehost.tld
1 2 * * * /path/to/your/command
```

Capture the output yourself

You can redirect stdout and stderr to a file. The exact syntax for capturing output may vary depending on what shell cron is using. Here are two examples which save all output to a file at `/tmp/mycommand.log` :

```
1 2 * * * /path/to/your/command &>/tmp/mycommand.log
1 2 * * * /path/to/your/command >/tmp/mycommand.log 2>&1
```

Look at the logs

Cron logs its actions via syslog, which (depending on your setup) often go to `/var/log/cron` or `/var/log/syslog` .

If required you can filter the cron statements with e.g.

```
grep CRON /var/log/syslog
```

Now that we've gone over the basics of cron, where the files are and how to use them let's look at some common problems.

Check that cron is running

If cron isn't running then your commands won't be scheduled ...

```
ps -ef | grep cron | grep -v grep
```

should get you something like

```
root    1224   1  0 Nov16 ?    00:00:03 cron
```

or

```
root    2018   1  0 Nov14 ?    00:00:06 crond
```

If not restart it

```
/sbin/service cron start
```

or

```
/sbin/service crond start
```

There may be other methods; use what your distro provides.

cron runs your command in a restricted environment.

What environment variables are available is likely to be very limited. Typically, you'll only get a few variables defined, such as `$LOGNAME`, `$HOME`, and `$PATH`.

Of particular note is the `PATH` is restricted to `/bin:/usr/bin`. **The vast majority of "my cron script doesn't work" problems are caused by this restrictive path**. If your command is in a different location you can solve this in a couple of ways:

1. Provide the full path to your command.

   ```
   1 2 * * * /path/to/your/command
   ```

2. Provide a suitable PATH in the crontab file

   ```
   PATH=/bin:/usr/bin:/path/to/something/else
   1 2 * * * command
   ```

If your command requires other environment variables you can define them in the crontab file too.

cron runs your command with cwd == $HOME

Regardless of where the program you execute resides on the filesystem, the current working directory of the program when cron runs it will be **the user's home directory**. If you access files in your program, you'll need to take this into account if you use relative paths, or (preferably) just use fully-qualified paths everywhere, and save everyone a whole lot of confusion.

The last command in my crontab doesn't run

Cron generally requires that commands are terminated with a new line. Edit your crontab; go to the end of the line which contains the last command and insert a new line (press enter).

Check the crontab format

You can't use a user crontab formatted crontab for /etc/crontab or the fragments in /etc/cron.d and vice versa. A user formatted crontab does not include a username in the 6th position of a row, while a system formatted crontab includes the username and runs the command as that user.

I put a file in /etc/cron.{hourly,daily,weekly,monthly} and it doesn't run

- Check that the filename doesn't have an extension see run-parts
- Ensure the file has execute permissions.
- Tell the system what to use when executing your script (eg. put `#!/bin/sh` at top)

Cron date related bugs

If your date is recently changed by a user or system update, timezone or other, then crontab will start behaving erratically and exhibit bizarre bugs, sometimes working, sometimes not. This is crontab's attempt to try to "do what you want" when the time changes out from underneath it. The "minute" field will become ineffective after the hour is changed. In this scenario, only asterisks would be accepted. Restart cron and try it again without connecting to the internet (so the date doesn't have a chance to reset to one of the time servers).

Percent signs, again

To emphasise the advice about percent signs, here's an example of what cron does with them:

```
# cron entry
* * * * * cat >$HOME/cron.out%foo%bar%baz
```

will create the ~/cron.out file containing the 3 lines

```
foo
bar
baz
```

This is particularly intrusive when using the `date` command. Be sure to escape the percent signs

```
* * * * * /path/to/command --day "$(date "+\%Y\%m\%d")"
```

How to use `sudo` in cron jobs

when running as a non-root user,

```
crontab -e
```

will open the user's crontab, while

```
sudo crontab -e
```

will open the root user's crontab. It's not recommended to run sudo commands in a cron job, so if you're trying to run a sudo command in a user's cron, try moving that command to root's cron and remove sudo from the command.

Share Improve this answer Follow

edited Jun 20, 2022 at 12:24

community wiki
34 revs, 16 users 53%
Eric Leschinski

May want to also mention in the 'restricted env' section that LD_LIBRARY_PATH may also need to have any additional directories set in case your cron task is failing on account of being unable to find shared libraries. – DavidJ Jun 23, 2015 at 13:10

1 The output capture does not work for me, may be because of the sh shell. I think this is more portable: `... /path/to/your/command >/tmp/mycommand.log 2>&1` – chus Aug 2, 2016 at 14:24

1 this worked for me: `sudo apt-get install postfix` – jmunsch Aug 2, 2017 at 3:07

3 This: `Cron generally requires that commands are terminated with a new line.` after spending a couple of hours figuring out why cron is not working. – slayedbylucifer May 11, 2022 at 10:56

1 Another possibility is the timezone difference. `date` command on my machine was showing UTC timezone while in crontab I was specifying local TZ. Worth checking if this is the issue. – tj-recess Aug 18, 2022 at 6:11

32

Debian Linux and its derivative (Ubuntu, Mint, etc) have some peculiarities that may prevent your cron jobs from executing; in particular, the files in `/etc/cron.d` , `/etc/cron.{hourly,daily,weekly,monthly}` must :

- be owned by root
- only be writable by root
- not be writable by group or other users
- have a name **without any dots '.'** or any other special character but '-' and '_' .

The last one hurts regularly unsuspecting users; in particular any script in one of these folders named `whatever.sh` , `mycron.py` , `testfile.pl` , etc. will **not** be executed, ever.

In my experience, this particular point has been by far the most frequent reason for a non-executing cronjob on Debian and derivatives.

See `man cron` for more details, if necessary.

Share Improve this answer Follow

answered Feb 4, 2016 at 20:29

wazoox
6,850 4 30 63

22

If your cronjobs stop working, check that your password hasnt expired., since once it has, all cron jobs stop. There will be messages in `/var/log/messages` similar to the one below which show issues with authenticating the user:

```
 (username) FAILED to authorize user with PAM (Authentication token is no longer valid; new
one required)
```

Share Improve this answer Follow

edited Oct 9, 2013 at 16:02
voretaq7
79.5k 17 129 214

answered Oct 9, 2013 at 15:29
Munkeh72
319 2 5

3 Just got this as well (error message file /var/log/syslog for me). In my case a DigitalOcean box that, at create time, they reset the root password (optionally) to another one, and apparently until you go in there and change it, all the cron jobs don't run. Bummer. Fix is something like `sudo -u root passwd` – rogerdpack Apr 1, 2016 at 16:13

17

Uncommon and irregular schedules

Cron is all things considered a very basic scheduler and the syntax does not easily allow an administrator to formulate slightly more uncommon schedules.

Consider the following job which commonly would be explained to *"run `command` every 5 minutes"*:

```
*/5 * * * * /path/to/your/command
```

versus:

```
*/7 * * * * /path/to/your/command
```

which **does not *always* run `command` every 7 minutes**.

Remember that the `/` character can be used to introduce a step but that steps don't wrap beyond the end of a series e.g. `*/7` which matches every 7th minute from the minutes `0-59` i.e. 0,7,14,21,28,35,42,49,56 but **between one hour and the next** there will be **only 4 minutes between batches**, after `00:56` a new series starts at `01:00`, `01:07` etc. (and batches won't run on `01:03`, `01:10`, `01:17` etc.).

What to do instead?

Create multiple batches

Rather than a single cron job, create multiple batches that combined result in the desired schedule.

For instance to run a batch every 40 minutes (00:00, 00:40, 01:20, 02:00 etc.) create two batches, one that runs twice on the even hours and second one that runs only the odd hours:

```
# The following lines create a batch that runs every 40 minutes i.e.

# runs on   0:00, 0:40,        02:00, 02:40         04:00 etc to 22:40
0,40 */2 * * * /path/to/your/command

# runs on               01:20,               03:20,       etc to 23:20
20 1/2 * * * /path/to/your/command

# Combined: 0:00, 0:40, 01:20, 02:00, 02:40, 03:20, 04:00 etc.
```

Run your batches less frequently

Rather than running your batch every 7 minutes, which is a difficult schedule to break down in multiple batches, simply run it every 10 minutes instead.

Start your batches more frequently (but prevent multiple batches from running concurrently)

Many odd schedules evolve because the batch runtimes increase/fluctuate and then the batches get scheduled with a bit of additional safety margin to prevent subsequent runs of the same batch from overlapping and running concurrently.

Instead, think differently and create a cronjob that will fail gracefully when a previous run has not finished yet, but which will run otherwise. See this Q&A:

```
* * * * * /usr/bin/flock -n /tmp/fcj.lockfile /usr/local/bin/frequent_cron_job
```

That will almost immediately start a new run once the previous run of /usr/local/bin/frequent_cron_job has completed.

Start your batches more frequently (but exit gracefully when the conditions are not right)

Since cron syntax is limited you may decide to **place more complex conditions and logic in the batch job itself** (or in a wrapper script around the existing batch job). That allows you to utilize the advanced capabilities of your favorite scripting languages, to comment your code and will prevent hard-to-read constructs in the crontab entry itself.

In bash the `seven-minute-job` would then look something like something like:

```
#!/bin/bash
# seven-minute-job
# This batch will only run when 420 seconds (7 min) have passed
# since the file /tmp/lastrun was either created or updated

if [ ! -f /tmp/lastrun ] ; then
    touch /tmp/lastrun
fi

if [ $(( $(date +%s) - $(date -r /tmp/lastrun +%s) )) -lt 420 ] ; then
     # The minimum interval of 7 minutes between successive batches hasn't passed yet.
    exit 0
fi
```

```
####  Start running your actual batch job below

/path/to/your/command

#### actual batch job is done, now update the time stamp
date > /tmp/lastrun
#EOF
```

Which you can then safely (attempt) to run every minute:

```
* * * * * /path/to/your/seven-minute-job
```

A different, but similar problem would to schedule a batch to run on the first Monday of every month (or the second Wednesday) etc. Simply schedule the batch to run every Monday and exit when date is neither between the 1st or 7th and the day of the week is not Monday.

```
#!/bin/bash
# first-monday-of-the-month-housekeeping-job

# exit if today is not a Monday (and prevent locale issues by using the day number)
if [ $(date +%u) != 1 ] ; then
  exit 0
fi

# exit if today is not the first Monday
if [ $(date +%d) -gt 7 ] ; then
  exit 0
fi

####  Start running your actual batch job below

/path/to/your/command

#EOF
```

Which you can then safely (attempt) to run every Monday:

```
0 0 * * 1 /path/to/your/first-monday-of-the-month-housekeeping-job
```

Don't use cron

If your needs are complex you might consider using a more advanced product that is designed to run complex schedules (distributed over multiple servers) and that supports triggers, job dependencies, error handling, retries and retry monitoring etc. The industry jargon would be "enterprise" job scheduling and/or "workload automation".

Share Improve this answer Follow

edited Jun 11, 2020 at 10:02
Community Bot
1

answered Nov 17, 2016 at 14:37
HBruijn
73.3k 23 130 194

"a more advanced product" is already available on most current linux systems: systemd timers. – azrdev Aug 26, 2022 at 19:19

8

PHP-specific

If you have some cron job like:

```
php /bla/bla/something.php >> /var/logs/somelog-for-stdout.log
```

And in case of errors expect, that they will be sent to you, but they not -- check this.

PHP by default not sending errors to STDOUT. @see https://bugs.php.net/bug.php?id=22839

To fix this, add in cli`s php.ini or in your line (or in your's bash wrapper for PHP) these:

- --define display_startup_errors=1
- --define display_errors='stderr'

1st setting will allow you to have fatals like 'Memory oops' and 2nd -- to redirect them all to STDERR. Only after you can sleep well as all will be sent to your root's mail instead of just logged.

Share Improve this answer Follow

answered Oct 23, 2013 at 4:45
gaRex
386 3 6

2 That error report was closed back in 2007 with the status of the patch being added to the PHP 5.2+ branches. Are you sure this is needed? I just tried on PHP 5.4 and it seems to work fine. (It is still needed for PHP 4 though). – Xeoncross Mar 5, 2014 at 20:34

@Xeoncross see date of answer :) – gaRex Mar 6, 2014 at 3:09

1 Yes, that is what confused me since you answered in 2013 and the ticket was back in '07. – Xeoncross Mar 6, 2014 at 4:55

4

Adding my answer from here for completeness, and adding another potentially helpful resource:

The `cron` user has a different `$PATH` than you do:

A frequent problem users make with `crontab` entries is that they forget that `cron` runs in a different `environment` than they do as a logged-in user. For example, a user creates a program or script in his `$HOME` directory, and enters the following command to run it:

```
$ ./certbot ...
```

The command runs perfectly from his command line. The user then adds that command to his `crontab`, but finds this does not work:

```
*/10 * * * * ./certbot ....
```

The reason for the failure in this case is that `./` is a different location for the `cron` user than it is for the logged-in user. That is, the `environment` is different! The PATH is part of the `environment`, and it is usually different for the `cron` user. Complicating this issue is that the `environment` for `cron` is not the same for all ***nix** distributions, and there are multiple versions of `cron`

A simple solution to this particular problem is to give the `cron` user a complete path specification in the `crontab` entry:

```
0 22 * * * /path/to/certbot .....
```

What is the `cron` user's `environment`?

In some instances, we may need to know the complete `environment` specification for `cron` on our system (or we may just be curious). **What is the `environment` for the `cron` user, and how is it different from ours?** Further, we may need to know the `environment` for another `cron` user - `root` for example... what is the `root` user's `environment` using `cron`? One way to learn this is to ask `cron` to tell us:

1. Create a shell script in your home directory (`~/`) as follows (or with the editor of your choice):

```
$ nano ~/envtst.sh
```

2. Enter the following in the editor, after adjusting for your system/user:

```
#!/bin/sh
/bin/echo "env report follows for user "$USER >> /home/you/envtst.sh.out
/usr/bin/env >> /home/you/envtst.sh.out
/bin/echo "env report for user "$USER" concluded" >> /home/you/envtst.sh.out
/bin/echo " " >> /home/you/envtst.sh.out
```

3. Save the file, exit the editor and set file permissions as executable.

```
$ chmod a+rx ~/envtst.sh
```

4. Run the script you just created, and review the output in `/home/you/envtst.sh.out`. This output will show your current environment as the `$USER` you're logged in as:

```
$ ./envtst.sh $$ cat /home/you/envtst.sh.out
```

5. Open your `crontab` for editing:

```
$ crontab -e -u root
```

6. Enter the following line at the bottom of your `crontab`:

```
* * * * *  /home/you/envtst.sh >> /home/you/envtst.sh.err 2>&1
```

ANSWER: The output file `/home/you/envtst.sh.out` will contain a listing of the `environment` for the "root cron user". Once you know that, adjust your `crontab` entry accordingly.

I can't specify the schedule I need in my `crontab` entry:

The schedule entry for `crontab` is of course defined in `man crontab`, and you should read this. However, reading `man crontab`, and understanding the schedule are two different things. And trial-and-error on a schedule specification can become *very* tedious. Fortunately, there is a resource that can help: the crontab guru.. Enter your schedule specification, and it will explain the schedule in plain English language.

Finally, and at risk of being redundant with one of the other answers here, do not get trapped into thinking that you are limited to a single `crontab` entry because you have one job to schedule. You are free to use as many `crontab` entries as you need to get the schedule you need.

Share Improve this answer Follow

answered Apr 9, 2019 at 16:48

Seamus
266 1 6

2 For me, the different environment has always caused issues with cron. This answer should get upvoted more.
– speedplane Nov 21, 2020 at 17:52

@speedplane: Couldn't agree more :) – Seamus Mar 29, 2022 at 16:48

1

As @Seamus said, 99% of my crontab problems come from having different environmental variables or different `PATH` variables so crontab can't find the script and it fails silently. Therefore my solution was to write a wrapper script that would:

1. Sets the `PATH` variable to the same one I'm used to
2. Sets my `PYTHONPATH` to my custom one to include all of my common functions
3. Changes the `DISPLAY` AND `DBUS` variables so that gui apps like notify-send messages actually make it to the screen
4. Changes the current working directory to the same one as the script. (If applicable)
5. Launches the script with `nice ionice -c3` to save on resources.
6. Logs the `stdout` and `stderr` of the script in the "logs" directory so that when something goes wrong I can actually find out what happened.

Now if I want to launch a script I just edit mycrontab with crontab -e to:

```
mm hh * * * /path-to.../cron.launcher.py script.name.sh
```

and it runs the script just as if I was trying to from the terminal without any aggravation. This was originally a BASH script, but it got hung up on arguments with spaces so I rewrote it all in python. It will even launch GUI apps into the userspace:

```
#!/usr/bin/python3

# Runtime directories:
SYSPATH = "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin"
PYTHONPATH = ""

################################################################################
import os
import time
import sys
import subprocess

# Get username passed by crontab
user = os.environ['LOGNAME']
uid = subprocess.check_output(('id -u ' + user).split()).decode().strip()
gid = subprocess.check_output(('id -g ' + user).split()).decode().strip()

# Set Environmental Variables:
os.environ["PATH"] = SYSPATH
os.environ["PYTHONPATH"] = PYTHONPATH
os.environ["DISPLAY"] = ":0"
os.environ["XAUTHORITY"] = os.path.join('/home', user, '.Xauthority')
os.environ["DBUS_SESSION_BUS_ADDRESS"] = 'unix:path=' + os.path.join('/run/user', uid, 'bus')
os.environ["XDG_RUNTIME_DIR"] = os.path.join("/run/user", uid)

# Get args:
script = sys.argv[1]
args = sys.argv[2:]
name = os.path.basename(script)
basedir = os.path.dirname(sys.argv[0])

# Log dir
logdir = os.path.join(basedir, 'cron.logs')
log = os.path.join(logdir, name + '.' + str(int(time.time())) + '.log')
os.makedirs(logdir, exist_ok=True)
if not os.access(logdir, os.W_OK):
    print("Can't write to log directory", logdir, file=sys.stderr)
    sys.exit(1)
```

Make sure you double check your `$PATH` and `$PYTHONPATH` variables in case you need to edit them to be different.

Share Improve this answer Follow

answered Apr 24, 2021 at 16:12

SurpriseDog
123 5

1 That was my case. Thanks for help! – sirajalam049 May 20, 2021 at 10:42

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.