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

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

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

which captures stdout and stderr to /tmp/mycommand.log

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=/usr:/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).