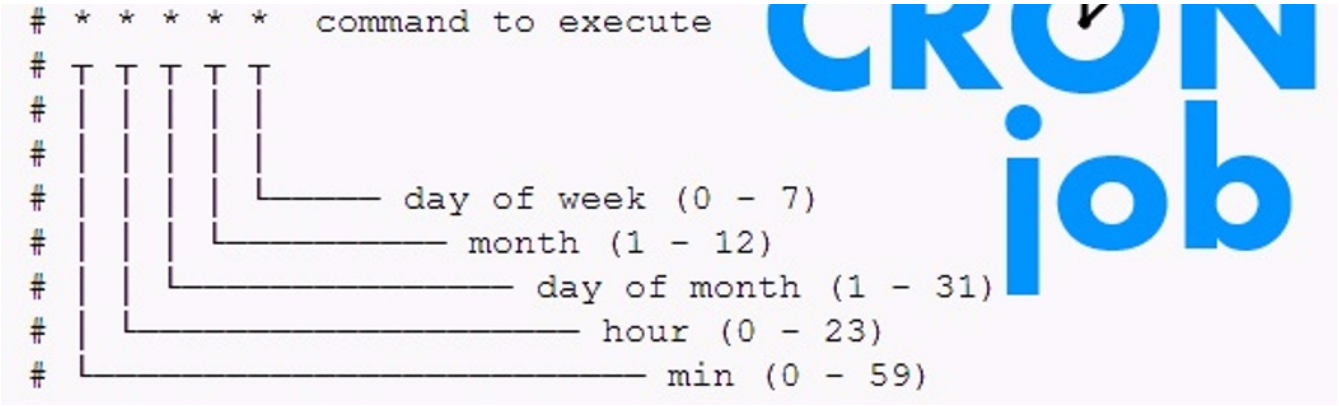
The Ultimate Crontab Cheatsheet



What is Crontab?

The Cron daemon is a service that runs on all main distributions of Unix and Linux. Specifically designed to execute commands at a given time. These jobs are commonly refered as cronjobs and are one of the essential tools that should be present in every Systems Administrator's tool box. Cronjobs are used for automating tasks or scripts so that they can be executed at specific time.

Creating a New Crontab

The crontab can be created easily by using the crontab command.

To Create or Edit your cron jobs

```
user@machine:~$ crontab -e
```

Crontab Commands

- `crontab -e` Edit or create a crontab file if doesn't already exist.
- `crontab -l` To Display the crontab file.
- `crontab -r` To Remove the crontab file.
- `crontab -v` To Display the last time you edited your crontab file. (This option is only available on a few systems.)

Crontab Parameters

```
# m h dom mon dow command
```

The above commented line is how the parameters of crontab are defined for each cronjob.

Available Crontab Fields

Field - Full Name - Allowed Values

- m - Minute - 0 to 59
- h - Hour - 0 to 23
- dom - Day of Month - 0 to 31
- mon - Month - 0 to 12
- dow - Day of Week - 0 to 7 (0 and 7 are both Sunday)

Or a better way to understand this is:

```
* * * * * command to be executed
- - - - -
| | | | |
| | | | +-- day of week (0 - 6) (Sunday=0)
| | | +---- month (1 - 12)
| | +------ day of month (1 - 31)
| +-------- hour (0 - 23)
+---------- min (0 - 59)
```

Names are allowed in some implementations of cron

These parameters are what allows the user to create scheduled jobs that run at a wide variety of times. The values allowed for each parameter provide the user with very fine details to manage the execution time. Special characters can also be used crontabs to more flexibility.

Special Characters

We can use these `Special characters` in cron to allow users to specify a time intervals in which the job should run. These special charaters can be used in crontabs for declaring the cronjobs.

Special Char: **Asterisk**

The Asterisk is the wild card character that is used to specify for all the occurrence of that parameter it is used for.

Ex:

```
* * * * * /home/user/script.sh
```

Special Char: **Comma**

The Comma is used when creating a list i.e. when we are declaring 2 or more execution times of a command.

Ex:

```
0,15,25 * * * * /home/user/script.sh
```

Special Char: **Hyphen**

The Hyphen is used to specify the range of time in which the script can run.

Ex:

```
0-59 0-23 * * * /home/user/script.sh
```

Special Char: **Forward Slash**

The Slash is used to create specified intervals of time within a range.

Ex:

```
*/20 * * * * /home/user/script.sh
```

Example Crontabs

Most of the crontabs can have multiple methods as they can be scheduled in many ways like using wild cards, or just defining the range.

Every Minute of Every Day

```
# m h dom mon dow command
 * * * * * /home/user/script.sh
```

or

```
# m h dom mon dow command
 0-59 0-23 0-31 0-12 0-7 /home/user/script.sh
```

Every 10 Minutes of Every Day

```
# m h dom mon dow command
 */10 * * * * /home/user/script.sh
```

or

```
# m h dom mon dow command
 0-59/10 * * * * /home/user/script.sh
```

or

```
# m h dom mon dow command
 0,10,20,30,40,50 * * * * /home/user/script.sh
```

Every 5 Minutes of the 6 am hour starting at 6:07

```
# m h dom mon dow command
07-59/5 06 * * * /home/user/script.sh
# This runs at 6:07, 6:012, 6:17, 6:22, 6:27, and so on until 6:57
```

Every day at midnight

```
# m h dom mon dow command
0 0 * * * /home/user/script.sh
```

or

```
# m h dom mon dow command
0 0 * * 0-7 /home/user/script.sh
```

Thrice Daily

```
# m h dom mon dow command
 0 */8 * * * /home/user/script.sh
```

or

```
# m h dom mon dow command
 0 0-23/8 * * * /home/user/script.sh
```

or

```
# m h dom mon dow command
 0 0,8,16 * * * /home/user/script.sh
```

Every weekday at 6 am

```
# m h dom mon dow command
 0 06 * * 1-5 /home/user/script.sh
```

Weekends at 6 am

```
# m h dom mon dow command
 0 06 * * 6,7 /home/user/script.sh
```

or

```
# m h dom mon dow command
 0 06 * * 6-7 /home/user/script.sh
```

Once a month on the 20th at 6 am

```
# m h dom mon dow command
 0 06 20 * * /home/user/script.sh
```

Every 4 days at 6 am

```
# m h dom mon dow command
 0 06 */4 * * /home/user/script.sh
```

Every 4 Months at 6 am on the 10th

```
# m h dom mon dow command
 0 06 10 */4 * /home/user/script.sh
```

Using special strings

Instead of typing those 5 parameters, we can save time and improve readability by using one of the 8 special strings:

Special String	Meaning
@reboot	Run once, at system startup
@yearly	Run once every year, “0 0 1 1 *”
@annually	(same as @yearly)
@monthly	Run once every month, “0 0 1 * *”
@weekly	Run once every week, “0 0 * * 0”
@daily	Run once each day, “0 0 * * *”
@midnight	(same as @daily)
@hourly	Run once an hour, “0 * * * *”

Example: special strings

Every hour

```
@hourly /home/user/script.sh
```

Every month

```
@monthly /home/user/script.sh
```

To Generate a log file

To store the cron output in a file, use the closing bracket (>) again:

```
10 * * * * /usr/bin/php /www/virtual/username/script.py > /var/log/cron.l
```

That will rewrite the output file every time. If you would like to append the output at the end of the file without a complete rewrite, use double closing bracket (>>) instead of single(>):

```
10 * * * * /usr/bin/php /www/virtual/username/script.py >> /var/log/cron.
```

To Block Output

If our script is very talkative, and prints all sort of information when it executes, we probably would want to silent these messages (unless we want those email messages). To do this, we should send all the normal output to a place called "/dev/null" which is basically like a black hole. It accepts anything you dump there, but we will never see it again.

Example:

```
# MAILTO=""
MAILTO=email@example.com
30 11 * * * /your/directory/whatever.py >/dev/null 2>&1
```

What is 2>&1 here?

Here,
File descriptor 1 is the standard output (stdout).
File descriptor 2 is the standard error (stderr).

```
echo test > afile.txt
```

..redirects stdout to afile.txt. This is the same as doing..

```
echo test 1> afile.txt
```

To redirect stderr, you do..

```
echo test 2> afile.txt
```

>& is the syntax used to redirect one stream to another file descriptor - 0 is for stdin. 1 is for stdout. 2 is for stderr.

You can redirect stdout to stderr by doing..

```
echo test 1>&2 # or echo test >&2
```

or vice versa:

```
echo test 2>&1
```

So, in short, 2> redirects stderr stream to an (unspecified) file, appending &1 redirects stderr to stdout stream.

Executable Scripts

```
10 * * * * /usr/bin/php /www/virtual/username/hello.php
```

Normally we would need to specify the parser at the beginning of the command as we have been doing. But there is actually a way to make our PHP scripts executable from the command line like a CGI script.

You need is to add the path to the parser as the first line of the script with shebang, like hello.php:

```
#!/usr/local/bin/php
<?php

echo "hello world\n";

// ...

?>
```

Also make sure to set proper permissions, chmod (like 755) to make the file executable.

```
chmod 755 hello.php
./hello.php
```

When you have an executable script, the cron job can be shorter like this:

```
10 * * * * /www/virtual/username/hello.php
```

What is #! in above scripts?

The **shebang line(#!)** in any script determines the script's ability to be executed like an standalone executable without typing python or php beforehand in the terminal or when double clicking it in a file manager(when configured properly). It isn't necessary but generally put there so when someone sees that file opened in a text editor, they will immediately know what type of script they are looking at. The correct usage of shebang is:

```
#!/usr/bin/env python
```

`#!/usr/bin/env` python Usually defaults to python 2.7.x, the latest version available on the system, and the following defaults to python 3.x, the latest.

```
#!/usr/bin/env python3
```

How to run multiple scripts at same time

you can use article "&" for parallel work and "&&" step-by-step launch script

```
# script.sh & script2.sh & script.sh
```