*The crontab is used to automate all types of tasks on Linux systems. This is an especially important skill for aspiring system administrators to learn. It can be somewhat challenging to get started if you’re a beginner. The syntax is different than most other commands. For this reason, this lesson will include a little more background information before I show you some of the uses.*

***Don’t be intimidated with crontab***

For me, Crontab was one of the more intimidating Linux concepts as a newbie. At the time that I was introduced to ‘crontab’, I had been using command line for only a few days and was barely understood how to use basic file navigation commands like ‘ls’ and ‘cd’.

The reason I bring up my personal learning history is that I expect a lot of new Linux users might feel similarly overwhelmed when they look at the unique syntax of ‘crontab’. I can reassure you though, it’s not that complicated once you understand how it works.

***What you’ll learn in this guide***

I want to give a quick introduction to some of the concepts involved with crontab to make it easier understand. My goal is to contextualize these concepts and illustrate how they relate to one another.

* Quick Introduction to Cron Concepts
* Setup Crontab Access for Your User Account
* Handle errors with your cronjobs
* Creating cronjobs

**Quick introduction to key cron concepts**

Let me first introduce you to some basic concept around ‘cron’.

**Difference between Cron, Crontab and Cron Job**

Seeing things visually helps understand new topics more quickly. Here is a breakdown of how these three topics generally interact. I will then describe each with more detail.

| **Element** | **Linux Name** | **Meaning** |
| --- | --- | --- |
| Daemon | ‘crond’ | Pronounced “demon” or “day-mon”. These are Linux background system processes. |
| Table | ‘crontab’ | You write rows to this table when entering a crontab command. Each ‘\*’ asterisk represents a segment of time and a corresponding column in each row. |
| Job | Cron Job | The specific task to be performed described in a row, paired with its designated time id |

**The Cron Table**

Crontab stands for Cron Table. This is a Linux system file that creates a table-like structure where fields are separated by white space. Users can populate the table by assigning values to each field (asterisk).

Throughout the article, I might use different language to describe this idea. To be clear a field, cell, column, etc. all refer to the same thing. If it helps, you can think of your crontab like a mini-database.

**The Cron Job**

If you’re not familiar with databases, you can imagine the cells in a blank Excel file. Either way, for this analogy each asterisk represents a column whose meaning is defined by its header. The final column will be a call for a command or script. Each complete row can be thought of as an individual job. These are often referred to as “cron jobs” although job, task, etc. are all interchangeable terms.

**The Cron Daemon**

We have already discussed the table and how we fill it with jobs. But, how do those jobs get executed? A system process called a Daemon [runs in the background of our Linux machine](https://linuxhandbook.com/run-process-background/).

There are Daemons for many different services. These are commonly named by suffixing a ‘d’ to a service name.

Naturally, the cron daemon is called ‘crond’. No action is required on our part to execute that daemon, but if you don’t think the command is working properly, you can use the [ps command](https://linuxhandbook.com/ps-command/) to verify that ‘crond’ is running.

ps aux | grep crond

This command will search current processes for all users and return any instances of ‘crond’.

christopher@pop-os:~$ ps ux | grep crond

christo+ 8942 0.0 0.0 18612 840 pts/0 S+ 02:16 0:00 grep --color=auto crond

I can see that the daemon is running for my user account. I already knew this because I have been populating a file all day long with output.

**Understanding Crontab syntax**

Now that you have a vague understanding of how cron works, let’s look at the syntax of using crontab. I hope it is less confusing if you can envision this information as a table in your mind.

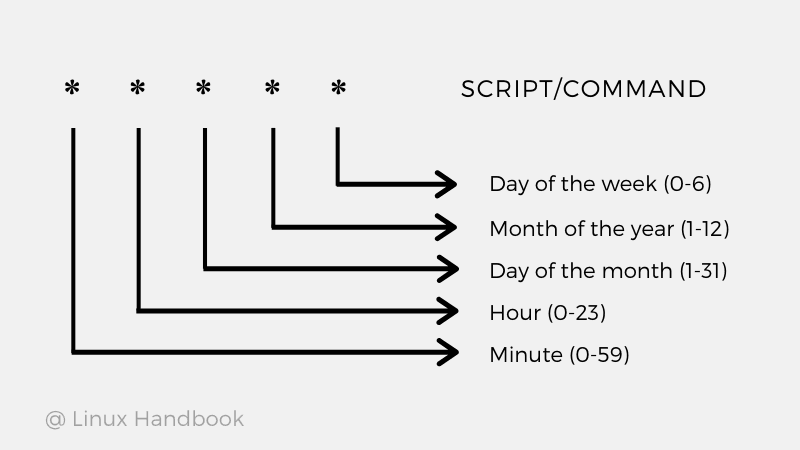
crontab [options]

\* \* \* \* \* <command>

OR

\* \* \* \* \* <path/to/script>

I promise this will make since to you once we get our own example up and running. Let’s go over the syntax for the cron jobs again.

[](https://linuxhandbook.com/content/images/2020/06/crontab-explanation.png)Crontab Explanation

As you can see, the crontab syntax has 5 asterisks. Here’s what each of those asterisk represent:

|  | **1st** | **2nd** | **3rd** | **4th** | **5th** |
| --- | --- | --- | --- | --- | --- |
|  | \* | \* | \* | \* | \* |
| ID | Minute | Hour | Day-Date | Month | Day Name |
| Values | 0-59 | 0 -23 | 1-31 | 1-12 | 0-6 |

NOTE: Day Names 0-6 begin with Sunday.

In order to schedule a task, you replace the appropriate asterisk with your desired value.

Let’s put it to some quick practice. If you have crontab like the one below, when do you think the job will run?

0 0 \* \* 0

**Q: When will the command be run if you set the job this way?**

A. Every hour from Monday thru Saturday  
B. Every minute on Sundays  
C. Only at midnight from Monday through Saturday  
D. Only at midnight on Sundays

The answer here is D. Run ‘command’ at 00:00 [midnight] every Sunday.

**Setup crontab cccess for your user account**

Crontab is user-specific. You already touched on that a little. If you think there’s a possibility you’ve already used crontab before, you can check that using **crontab -l**.

christopher@pop-os:~$ crontab -l

no crontab for christopher

christopher@pop-os:~$ crontab -e

no crontab for christopher - using an empty one

Select an editor. To change later, run 'select-editor'.

1. /bin/nano <---- easiest

2. /usr/bin/code

3. /bin/ed

Choose 1-3 [1]: 1

When I run this command, you can see that I have no crontab on this system.

Since I haven’t created a crontab yet, when I use -e to edit the table, it prompts me for my preferred text editor. [Nano](https://www.nano-editor.org/) is suggested as the easiest program to use. You can use any command line text editor like Vim or Emacs. It’s really up to you.

**If At First You Don’t Succeed, Sudo**

If you attempt the **crontab -e** command but don’t get this result, you may not have the user privileges to create the table. If you have sudo access, you can use sudo to set the crontab.

sudo crontab -e <your\_username>

Your system will automatically load your crontab to the proper destination which can vary based on distribution but is often in a directory like **/var/spool/cron/crontabs**. Do not attempt to edit files here.

**What happens if the cron job encounters an error?**

The default behavior is to email it’s output. This feature is designed for admins who can automatically have logs sent to a “local” email on the network domain.

You can set this up yourself if you have a mail server. There are also ways to automate email output to GMail or similar services. However, those methods are beyond the scope of this article.

Instead, we will look at two common ways around the error.

**1) Send Output to a File**

You can designate a file for this type of output to be sent to and then use >> to redirect the output.

Using >> will append information to an existing file, while a single > symbol will overwrite the file. This is important to know if you want to maintain a large log file that updates records frequently. Both will automatically create the file if it does not exist.

Cron Job Example:

0 \* \* \* \* echo "Linux is Cool!" >> ~/crontab\_log.txt

**2) Use /dev/null**

This will bypass the email option by in essence deleting the data. Standard error (‘2’) and standard output (‘1’) are both sent to the null file.

0 0 \* \* \* echo "Why are you silencing me every night at midnight?" > /dev/null 2>&1

You might have noticed I am using echo commands for the examples. There’s no particular reason for that, but it makes it easy to verify changes and “check your work”.

If you’ve done some programming, you may have used print commands to test your logic. This is the same concept.

Let’s try to set up our own cron job. If you’ve already been “playing along”, that’s great. If not, now’s the time to get that terminal ready and have some fun.

**Crontab examples: scheduling commands and scripts**

I showed you a couple examples while I was explaining how output gets routed. Did those make sense to you?

Let me walk through the first example.

| **Minute** | **Hour** | **Day-Date** | **Month** | **Day Name** | **Command** |
| --- | --- | --- | --- | --- | --- |
| 0 | \* | \* | \* | \* | echo “Linux is Cool!” >> ~/crontab\_log.txt |

Setting the minute value to ‘0’ means the command will run every hour, on the hour.

**Advanced Job Schedules**

You can edit multiple values at once. If you want, you can replace all 5 of the asterisks with specifications.

| **Minute** | **Hour** | **Day-Date** | **Month** | **Day Name** | **Command** |
| --- | --- | --- | --- | --- | --- |
| \*/5 | 3-6 | \*/5 | \*/2 | 0,6 | echo “Linux is Cool!” >> ~/crontab\_log.txt |

Any idea what this one says? For the sake of this tutorial, I made this job particularly confusing. It would be unusual to have something with so many parameters “in the wild”, but let’s see if you can decipher it. For something like this, I like to work backwards through the fields.

Let’s try that together:

| **Field** | **Value** | **Meaning** |
| --- | --- | --- |
| Day Name | 0,6 | Saturday and Sunday |
| Month | \*/2 | Every month that is divisible by 2 (even) months. |
| Day Date | \* | Every Date |
| Hour | 3-6 | Between 3 and 6 AM |
| Minutes | \*/5 | Every 5 Minutes |

In Plain Language:

***So every other month, on weekends, regardless of the date, this command will run every 5 minutes between 3 am and 6 am.***

Wow, that was convoluted. If you were able to follow that, you are prepared to cron job with the best of them.

**Write a simple cron automation script for backing up files**

Up until this point, the cron jobs you’ve written have done just one thing. That can be useful, but maybe you want to do multiple tasks.

Fortunately this is not only possible but also very easy. If you remember from the original syntax example, you can also use a path to a script.

This isn’t limited to just bash either, you can also implement a script that uses Python or [Perl](https://www.perl.org/) if you want.

**What are our goals?**

* Jobs will be processed at 3 AM each night
* Back up /Documents folder to a zip file
* Generate a text file with a list of everything in the directory
* Create an archive folder that that clones our backup and text file into a sub-folder with the current date

our\_backup\_script.sh

#! /bin/bash

DATE=$(date +%d-%m-%Y)

# Date in format DAY##-MONTH##-YEAR####

mkdir -p ~/archive/$DATE

# create a folder for today's date in the archive, if archive doesn't exist, make archive

ls -al ~/Documents > ~/archive/$DATE/contents.txt

# create a text file listing the contents of the documents folder

cd ~/ && tar -cpzf $DATE.docs.backup.gz Documents/\*

# change to parent directory to tar /Documents folder

cp ~/$DATE.docs.backup.gz ~/archive/$DATE/documents\_archive.gz

# one .gz file is left in the home directory, a clone is sent to our archive under it's date

christopher@pop-os:~$ ls

Desktop Downloads Music Pictures Public Videos

Documents ENV our\_backup\_script.sh projects Templates 'VirtualBox VMs'

christopher@pop-os:~$ bash our\_backup\_script.sh

christopher@pop-os:~$ ls

25-11-2019.docs.backup.gz Documents Music projects Videos

archive Downloads our\_backup\_script.sh Public 'VirtualBox VMs'

Desktop ENV Pictures Templates

christopher@pop-os:~$ ls archive/25-11-2019/

contents.all\_files.txt documents\_archive.gz

All that’s left to do make this script a cron job.

crontab -e

And add the following in there:

0 3 \* \* \* bash ~/our\_backup\_script.sh

Was yours successful? Mine was. In fact, I liked the idea so much, that I might just keep this as a daily backup. One modification I will make is to move my archive to a folder on my machine that syncs with cloud storage.

Here's a quick summary of what you learned:

Another way to [schedule jobs in Linux is using at command](https://linuxhandbook.com/at-command/). You might be interested in learning about that as well.

Do you have any ideas for a script you might want to create? Did this article help you understand crontab better? Share with us in the comments.