

crontab Script Scheduling

Contents

crontab Script Scheduling.....	1
What Is a Cron Job?	1
Create or Edit a crontab File	2
Cron Syntax	3
Cron Job Special Strings.....	5
Tutorial 1: Create a Cron Job.....	5
Create cron job.....	6
Assignment Submission.....	8

Time required: 30 minutes

What Is a Cron Job?

Cron is a Linux utility program that lets users input commands for scheduling tasks repeatedly at a specific time. Tasks scheduled in cron are called cron jobs. Users can determine what kind of task they want to automate and when it should be executed.

Cron is a daemon – a background process executing non-interactive jobs. In Windows, you might be familiar with background processes such as the Task Scheduler that work similarly to the cron daemon.

A daemon is always idle, waiting for a command to request it to perform a particular task.

A cron file is a simple text file that contains commands to run periodically at a specific time. The default system cron table or crontab configuration file is `/etc/crontab`, located within the crontab directory `/etc/cron.*`.

Only system administrators can edit the system crontab file. Unix-like operating systems support multiple admins. Each can create a crontab file and write commands to perform jobs anytime they want.

With cron jobs, users can automate system maintenance, disk space monitoring, and schedule backups. Because of their nature, cron jobs are great for computers that work 24/7, such as servers.

While cron jobs are used mainly by system administrators, they can be beneficial for web developers too.

For instance, as a website administrator, you can set up one cron job to automatically backup your site every day at midnight, another to check for broken links every Monday at midnight, and a third to clear your site cache every Friday at noon.

However, like any other program, cron has limitations you should consider before using it:

- The shortest interval between jobs is 60 seconds. With cron, you won't be able to repeat a job every 59 seconds or less.
- Centralized on one computer. Cron jobs can't be distributed to multiple computers on a network. If the computer running cron crashes, the scheduled tasks won't be executed, and the missed jobs will only be able to be run manually.
- No auto-retry mechanism. Cron is designed to run at strictly specified times. If a task fails, it won't run again until the next scheduled time. This makes cron unsuitable for incremental tasks.

With these limitations, cron is an excellent solution for simple tasks that run at a specific time with regular intervals of at least 60 seconds.

Before creating a Cron Job, make sure that your script works. To do that, open the file in your browser (by URL) or execute it via SSH, depending on what type of script you have.

Before proceeding with the basic operations of cron, it's essential to know the different cron job configuration files:

- The system crontab. Use it to schedule system-wide, essential jobs that can only be changed with root privileges.
- The user crontab. This file lets users create and edit cron jobs that only apply at the user level.

Create or Edit a crontab File

To see a list of active scheduled tasks in your system, enter the following command:

```
crontab -l
```

To create or edit a crontab file, enter the following into the command line:

```
crontab -e
```

If no crontab files are found in your system, the command will automatically create a new one. `crontab -e` allows you to add, edit, and delete cron jobs.

You'll need a text editor like `vi` or `nano` to edit a crontab file. When entering `crontab -e` for the first time, you'll be asked to choose which text editor you want to edit the file with.

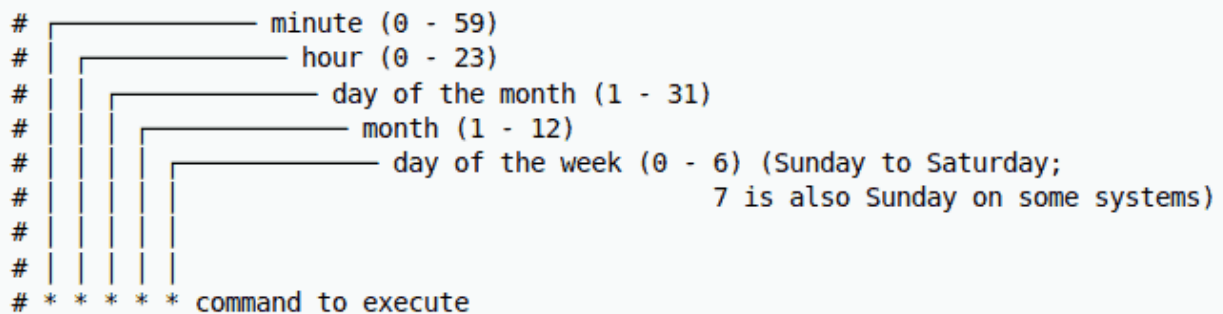
To delete all scheduled tasks in your crontab file and start fresh, type the following command:

```
crontab -r
```

This command will prompt the user with a yes/no option before removing the crontab file.

Cron Syntax

Let's look at cron's syntax and formatting.



```
# minute (0 - 59)
# hour (0 - 23)
# day of the month (1 - 31)
# month (1 - 12)
# day of the week (0 - 6) (Sunday to Saturday;
#                               7 is also Sunday on some systems)
#
# * * * * * command to execute
```

The diagram shows a crontab entry with five fields separated by spaces. Each field is labeled with a line to its right, connected by a vertical line. The labels are: minute (0 - 59), hour (0 - 23), day of the month (1 - 31), month (1 - 12), and day of the week (0 - 6) (Sunday to Saturday; 7 is also Sunday on some systems). The final line shows the asterisk notation for each field followed by the command to execute.

The crontab syntax consists of five fields with the following possible values:

- **Minute:** The minute of the hour the command will run on, ranging from 0-59.
- **Hour:** The hour the command will run at, ranging from 0-23 in the 24-hour notation.
- **Day of the month:** The day of the month the user wants the command to run on, ranging from 1-31.
- **Month:** The month that the user wants the command to run in, ranging from 1-12, thus representing January-December.
- **Day of the week:** The day of the week for a command to run on, ranging from 0-6, representing Sunday-Saturday. In some systems, the value 7 represents Sunday.

Don't leave any of the fields blank.

If you want to set up a cron job to run root/backup.sh every Friday at 5:37 pm, here's what your cron command should look like:

```
37 17 * * * root/backup.sh
```

In the example above, 37 and 17 represent 5:37 pm. Both asterisks for the Day of the month and Month fields signify all possible values. This means that the task should be repeated no matter the date or the month. Finally, 5 represents Friday. The set of numbers is then followed by the location of the task itself.

If you're not sure about manually writing the cron syntax, you can use free tools like the following to generate the exact numbers for the time and date you want for your command.

- [Crontab Generator](#)

To set the correct time for your cron command, knowledge of cron job operators is essential. They allow you to specify which values you want to enter in each field. You need to use proper operators in all crontab files.

- **Asterisk (*)** Use this operator to signify all possible values in a field. For example, if you want your cron job to run every minute, write an asterisk in the Minute field.
- **Comma (,)** Use this operator to list multiple values. For example, writing 1,5 in the Day of the week field will schedule the task to be performed every Monday and Friday.
- **Hyphen (-)** Use this operator to determine a range of values. For example, if you want to set up a cron job from June to September, writing 6-9 in the Month field will do the job.
- **Separator (/)** Use this operator to divide a value. For example, if you want to make a script run every twelve hours, write */12 in the Hour field.
- **Last (L)** This operator can be used in the day-of-month and day-of-week fields. For example, writing 3L in the day-of-week field means the last Wednesday of a month.
- **Weekday (W)** Use this operator to determine the closest weekday from a given time. For example, if the 1st of a month is a Saturday, writing 1W in the day-of-month field will run the command on the following Monday (the 3rd).
- **Hash (#)** Use this operator to determine the day of the week, followed by a number ranging from 1 to 5. For example, 1#2 means the second Monday of the month.
- **Question mark (?)** Use this operator to input "no specific value" for the "day of the month" and "day of the week" fields.

Cron Job Special Strings

Special strings are used to schedule cron jobs at time intervals without the user having to figure out the logical set of numbers to input. To use them, write an @ followed by a simple phrase.

Here are some useful special strings that you can use in commands:

- **@hourly** The job will run once an hour.
- **@daily** or **@midnight** These strings will run the task every day at midnight.
- **@weekly** Use this to run jobs once a week at midnight on Sunday.
- **@monthly** This special string runs a command once on the first day of every month.
- **@yearly** Use this to run a task once a year at midnight on January 1st.
- **@reboot** With this string, the job will run only once at startup.

Tutorial 1: Create a Cron Job

This tutorial is to be done at a terminal prompt logged in as user.

1. Run this command to list current running daemons.

```
systemctl --type=service --state=running
```

2. Run this command to list active scheduled tasks in your system.

```
crontab -l
```

3. Type **nano my_script.sh**
4. This will create an empty text file. This will be our test script.
5. Enter the following script:

```
1  #!/bin/bash
2  # shebang, tells the script where the interpreter is
3  # Make the script executable: sudo chmod 744
4
5  # echo normally prints to console
6  # >> redirects and append output to the logfile
7  echo "Hello there, $(whoami)!" >> logfile
8  echo "Today is $(date)" >> logfile
9  ls >> logfile
```

6. Save the file (CTRL S) and exit nano (CTRL X).
7. Let's test the script. **bash ./my_script.sh**
8. There shouldn't be any output to the console. We redirected the output of the ls command to logfile.
9. Type: **cat logfile**
10. You should see the results of the commands in the logfile.
11. Insert a screenshot showing the contents of the cat command.

Click or tap here to enter text.

Let's change the permissions to make this file executable for the current user.

12. Type: **chmod 744 my_script.sh**
13. Type: **ls -l**
14. You should see that all permissions have been given to the user.
my_script.sh should be green to indicate it is an executable script.
15. Insert a screenshot of the results of your **ls -l** command.

Click or tap here to enter text.

Let's test our script to make sure it works. Let's delete the logfile to make it easier to see that our script worked.

16. Type: **rm logfile**
17. Type: **./my_script.sh**
18. Type: **cat logfile**
19. Insert a screenshot of the results of your ls -l command.

Click or tap here to enter text.

Create cron job

We are going to schedule our script to run every 60 seconds. This will allow us to see if it works properly.

1. Go to: <http://www.crontab-generator.org>

2. Leave all the settings as they are. We want to schedule our crontab job every minute.
3. Go down to where it says: **Command To Execute**
4. Type in: **/home/user/my_script.sh**
5. Go to the bottom: Click **Generate Crontab Line**
6. The result will appear at the top of the page.

```
Cron Job Generated (you may copy & paste it to your crontab):

***** /home/user/my_script.sh >/dev/null 2>&1

Your cron job will be run at: (5 times displayed)

• 2025-04-02 00:56:00 UTC
• 2025-04-02 00:57:00 UTC
• 2025-04-02 00:58:00 UTC
• 2025-04-02 00:59:00 UTC
• 2025-04-02 01:00:00 UTC
• ...
```

7. At a terminal prompt → **crontab -e**
8. Choose 1 for nano.
9. Go to the bottom of the script.
10. Copy and paste the crontab line that you generated earlier into nano.
11. Save the file and quit nano.
12. You should see the following if you were successful.

```
(user@kali)-[~]
$ crontab -e
no crontab for user - using an empty one
Select an editor. To change later, run select-editor again.
 1. /bin/nano          ← easiest
 2. /usr/bin/vim.basic
 3. /usr/bin/vim.tiny

Choose 1-3 [1]: 1
crontab: installing new crontab
```

13. It may take a minute for the job to run.

14. Type: **ls -l**

15. You should see that the time on the logfile changed.

16. Insert two screenshots a minute apart to show that your crontab job is running.

[Click or tap here to enter text.](#)

17. Let's remove the current crontab file. We really don't need this job running every minute.

18. Type: **crontab -r**

Assignment Submission

Attach this completed document to the assignment in BlackBoard.