How to Automate Tasks with cron Jobs in Linux

If you're working in IT, you might need to schedule various repetitive tasks as part of your automation processes.

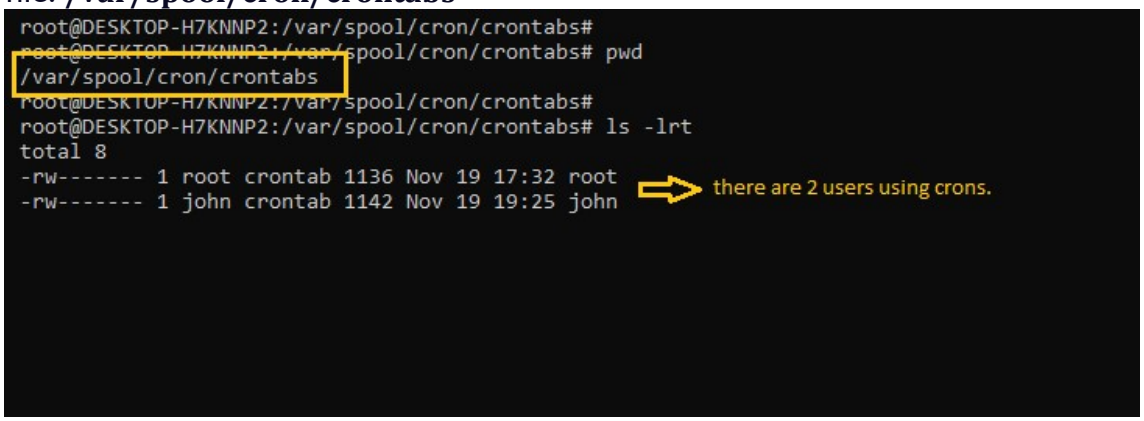
For example, you could schedule a particular job to periodically execute at specific times of the day. This is helpful for performing daily backups, monthly log archiving, weekly file deletion to create space, and so on.

And if you use Linux as your OS, you'll use something called a cron job to make this happen.

What is a cron?

Cron is a job scheduling utility present in Unix like systems. The crond daemon enables cron functionality and runs in background. The cron reads the **crontab** (cron tables) for running predefined scripts.
By using a specific syntax, you can configure a cron job to schedule scripts or other commands to run automatically.

For individual users, the cron service checks the following file: **/var/spool/cron/crontabs**



Contents of /var/spool/cron/crontabs

What are cron jobs in Linux?

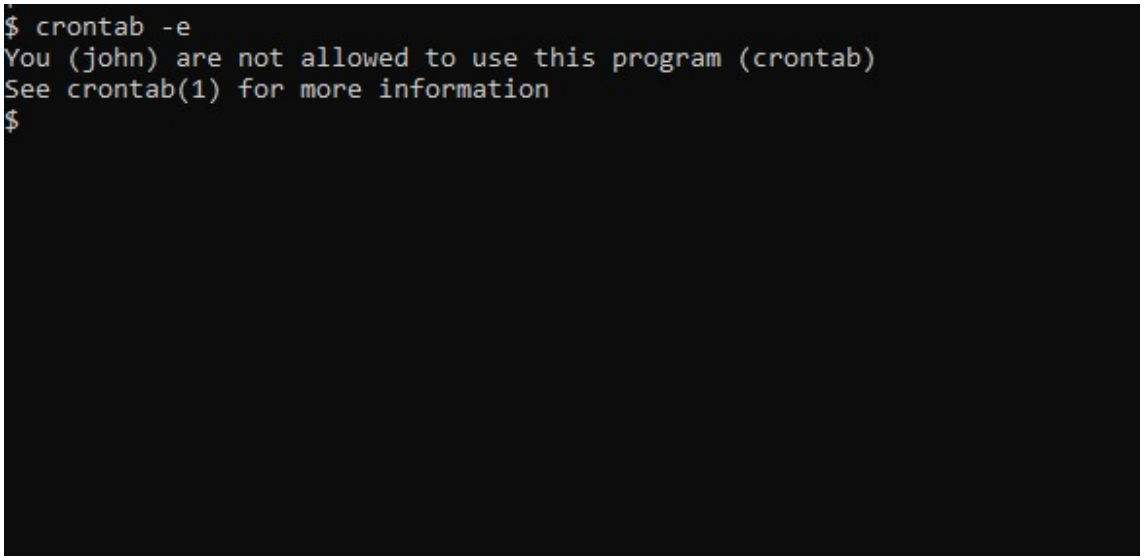
Any task that you schedule through crons is called a cron job. Cron jobs help us automate our routine tasks, whether they're hourly, daily, monthly, or yearly.

Now, let's see how cron jobs work.

How to Control Access to crons

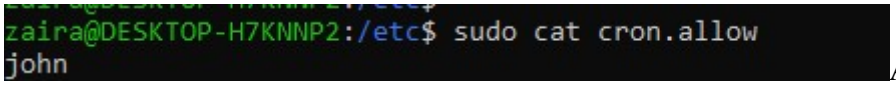
In order to use cron jobs, an admin needs to allow cron jobs to be added for users in the '/etc/cron.allow' file.

If you get a prompt like this, it means you don't have permission to use cron.



Cron job addition denied for user John.

To allow John to use crons, include his name in '/etc/cron.allow'. This will allow John to create and edit cron jobs.



Allowing John in file cron.allow

Users can also be denied access to cron job access by entering their usernames in the file '/etc/cron.d/cron.deny'.

How to Add cron Jobs in Linux

First, to use cron jobs, you'll need to check the status of the cron service. If cron is not installed, you can easily download it through the package manager. Just use this to check:

```
# Check cron service on Linux system
sudo systemctl status cron.service
```

Cron job syntax

Crontabs use the following flags for adding and listing cron jobs.

- **crontab -e**: edits crontab entries to add, delete, or edit cron jobs.
- **crontab -l**: list all the cron jobs for the current user.
- **crontab -u username -l**: list another user's crons.
- **crontab -u username -e**: edit another user's crons.

When you list crons, you'll see something like this:

```
# Cron job example
```

```
* * * * * sh /path/to/script.sh
```

In the above example,

- * * * * * represents minute(s) hour(s) day(s) month(s) weekday(s), respectively.
-

	VALUE	DESCRIPTION
Minutes	0-59	Command would be executed at the specific minute.
Hours	0-23	Command would be executed at the specific hour.
Days	1-31	Commands would be executed in these days of the months.
Months	1-12	The month in which tasks need to be executed.
Weekdays	0-6	Days of the week where commands would run. Here, 0 is Sunday.

- `sh` represents that the script is a bash script and should be run from `/bin/bash`.
- `/path/to/script.sh` specifies the path to script.

Below is the summary of the cron job syntax.

```
*   *   *   *   *  sh /path/to/script/script.sh
|   |   |   |   |              |
|   |   |   |   |      Command or Script to Execute
|   |   |   |   |
|   |   |   |   |
|   |   |   |   |
|   |   |   | Day of the Week(0-6)
|   |   |   |
|   |   | Month of the Year(1-12)
|   |   |
|   | Day of the Month(1-31)
|   |
| Hour(0-23)
|
Min(0-59)
```

Cron job examples

Below are some examples of scheduling cron jobs.

SCHEDULE	SCHEDULED VALUE
5 0 * 8 *	At 00:05 in August.
5 4 * * 6	At 04:05 on Saturday.
0 22 * * 1-5	At 22:00 on every day-of-week from Monday through Friday.

It is okay if you are unable to grasp this all at once. You can practice and generate cron schedules with the crontab guru.

How to set up a cron job

In this section, we will look at an example of how to schedule a simple script with a cron job.

1. Create a script called `date-script.sh` which prints the system date and time and appends it to a file. The script is shown below:

```
root@osboxes:~#
root@osboxes:~# cat date-script.sh
#!/bin/sh

echo `date` >> date-out.txt
root@osboxes:~#
```

Script for printing date.

2. Make the script executable by giving it execution rights.

```
chmod 775 date-script.sh
```

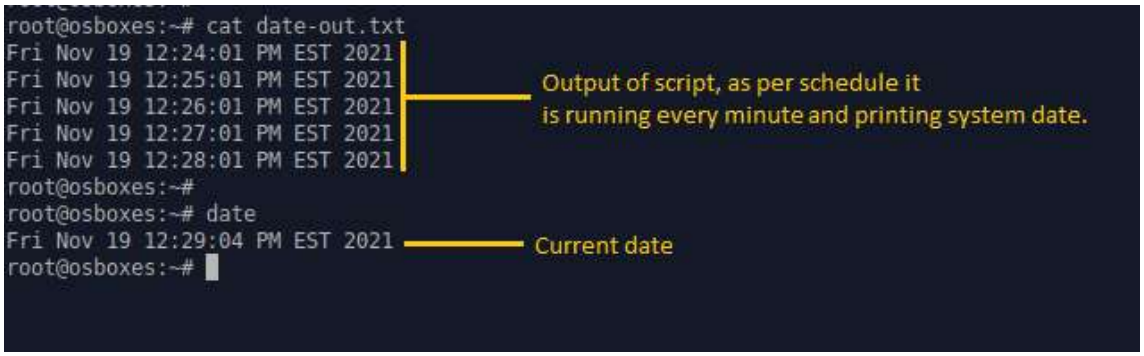
3. Add the script in the crontab using `crontab -e`.

Here, we have scheduled it to run per minute.

```
*/1 * * * * /bin/sh /root/date-script.sh
~
~
~
~
~
```

Adding a cron job in crontab every minute.

4. Check the output of the file `date-out.txt`. According to the script, the system date should be printed to this file every minute.



Output of our cron job.

How to Troubleshoot crons

Crons are really helpful, but they might not always work as intended. Fortunately, there are some effective methods you can use to troubleshoot them.

1. **Check the schedule.**

First, you can try verifying the schedule that's set for the cron. You can do that with the syntax you saw in the above sections.

2. **Check cron logs.**

First you need to check if the cron has run at the intended time or not. You can verify this from the cron logs located at `var/log/cron`. In some distros, logs can be found at `/var/log/syslog`

If there is an entry in these logs at the correct time, it means the cron has run according to the schedule you set.

Below are the logs of our cron job example. Note the first column which shows the timestamp. The path of the script is also mentioned at the end of the line.

```
Nov 19 12:23:01 osboxes CRON[4169]: (osboxes) CMD (/bin/bash /home/osboxes/cron-ex/sample.sh)
Nov 19 12:23:01 osboxes CRON[4170]: (root) CMD (/bin/sh /root/date-script.sh)
Nov 19 12:24:01 osboxes CRON[4179]: (root) CMD (/bin/sh /root/date-script.sh)
Nov 19 12:24:01 osboxes CRON[4180]: (osboxes) CMD (/bin/bash /home/osboxes/cron-ex/sample.sh)
Nov 19 12:25:01 osboxes CRON[4190]: (osboxes) CMD (/bin/bash /home/osboxes/cron-ex/sample.sh)
Nov 19 12:25:01 osboxes CRON[4191]: (root) CMD (/bin/sh /root/date-script.sh)
Nov 19 12:25:59 osboxes crontab[4198]: (root) BEGIN EDIT (root)
Nov 19 12:26:01 osboxes CRON[4205]: (root) CMD (/bin/sh /root/date-script.sh)
Nov 19 12:26:01 osboxes CRON[4206]: (osboxes) CMD (/bin/bash /home/osboxes/cron-ex/sample.sh)
Nov 19 12:27:01 osboxes CRON[4213]: (osboxes) CMD (/bin/bash /home/osboxes/cron-ex/sample.sh)
Nov 19 12:27:01 osboxes CRON[4214]: (root) CMD (/bin/sh /root/date-script.sh)
Nov 19 12:28:01 osboxes CRON[4221]: (osboxes) CMD (/bin/bash /home/osboxes/cron-ex/sample.sh)
Nov 19 12:28:01 osboxes CRON[4222]: (root) CMD (/bin/sh /root/date-script.sh)
Nov 19 12:28:47 osboxes crontab[4198]: (root) END EDIT (root)
Nov 19 12:29:01 osboxes CRON[4231]: (osboxes) CMD (/bin/bash /home/osboxes/cron-ex/sample.sh)
Nov 19 12:29:01 osboxes CRON[4232]: (root) CMD (/bin/sh /root/date-script.sh)
```

Cron job logs.

3. Redirect cron output to a file.

You can redirect a cron's output to a file and check the file for any possible errors.

```
# Redirect cron output to a file
* * * * * sh /path/to/script.sh &> log_file.log
```

Wrapping up

Automating tasks, like with cron jobs, reduces the repetitive work you need to do. It also lets machines auto heal and work around the clock without human intervention.

Automation in Linux heavily relies on cron jobs, so you should definitely learn crons and experiment with them.