**How to use cron in Linux**

**Common (and uncommon) cron uses**

I use the **cron** service to schedule obvious things, such as regular backups that occur daily at 2 a.m. I also use it for less obvious things.

* The system times (i.e., the operating system time) on my many computers are set using the Network Time Protocol (NTP). While NTP sets the system time, it does not set the hardware time, which can drift. I use cron to set the hardware time based on the system time.
* I also have a Bash program I run early every morning that creates a new "message of the day" (MOTD) on each computer. It contains information, such as disk usage, that should be current in order to be useful.
* Many system processes and services, like [Logwatch](https://sourceforge.net/projects/logwatch/files/" \t "_blank), [logrotate](https://github.com/logrotate/logrotate" \t "_blank), and [Rootkit Hunter](http://rkhunter.sourceforge.net/), use the cron service to schedule tasks and run programs every day.

The **crond** daemon is the background service that enables cron functionality.

The cron service checks for files in:

* **/var/spool/cron**
* **/etc/cron.d** directories
* **/etc/anacrontab** file

The contents of these files define cron jobs to be run at various intervals.

The individual user cron files are located in **/var/spool/cron**

system services and applications generally add cron job files in the **/etc/cron.d** directory

**/etc/anacrontab** is a special case that will be covered later in this article.

**Using crontab**

The cron utility runs based on commands specified in a cron table (**crontab**). Each user, including root, can have a cron file. These files don't exist by default, but can be created in the **/var/spool/cron** directory using the **crontab -e** command that's also used to edit a cron file (see the script below). I strongly recommend that you *not* use a standard editor. Using the **crontab** command not only allows you to edit the command, it also restarts the **crond** daemon when you save and exit the editor. The **crontab** command uses Vi as its underlying editor.

New cron files are empty, so commands must be added from scratch. I added the job definition example below to my own cron files, just as a quick reference, so I know what the various parts of a command mean. Feel free to copy it for your own use.

***# crontab -e*  
SHELL=/bin/bash  
MAILTO=root@example.com  
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin  
  
*# For details see man 4 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) OR sun,mon,tue,wed,thu,fri,sat*  
*# |  |  |  |  |*  
*# \*  \*  \*  \*  \* user-name  command to be executed*  
  
*# backup using the rsbu program to the internal 4TB HDD and then 4TB external*  
01 01 \* \* \* /usr/local/bin/rsbu -vbd1 ; /usr/local/bin/rsbu -vbd2  
  
*# Set the hardware clock to keep it in sync with the more accurate system clock*  
03 05 \* \* \* /sbin/hwclock --systohc  
  
*# Perform monthly updates on the first of the month*  
*# 25 04 1 \* \* /usr/bin/dnf -y update***

The**crontab**command is used to view or edit the cron files.

The first three lines in the code above set up a default environment. The environment must be set to whatever is necessary for a given user because cron does not provide an environment of any kind. The **SHELL** variable specifies the shell to use when commands are executed. This example specifies the Bash shell. The **MAILTO** variable sets the email address where cron job results will be sent. These emails can provide the status of the cron job (backups, updates, etc.) and consist of the output you would see if you ran the program manually from the command line. The third line sets up the **PATH** for the environment. Even though the path is set here, I always prepend the fully qualified path to each executable.

There are several comment lines in the example above that detail the syntax required to define a cron job. I'll break those commands down, then add a few more to show you some more advanced capabilities of crontab files.

**01 01 \* \* \* /usr/local/bin/rsbu -vbd1 ; /usr/local/bin/rsbu -vbd2**

This line in my**/etc/crontab**runs a script that performs backups for my systems.

This line runs my self-written Bash shell script, **rsbu**, that backs up all my systems. This job kicks off at 1:01 a.m. (01 01) every day. The asterisks (\*) in positions three, four, and five of the time specification are like file globs, or wildcards, for other time divisions; they specify "every day of the month," "every month," and "every day of the week." This line runs my backups twice; one backs up to an internal dedicated backup hard drive, and the other backs up to an external USB drive that I can take to the safe deposit box.

The following line sets the hardware clock on the computer using the system clock as the source of an accurate time. This line is set to run at 5:03 a.m. (03 05) every day.

**03 05 \* \* \* /sbin/hwclock --systohc**

This line sets the hardware clock using the system time as the source.

I was using the third and final cron job (commented out) to perform a **dnf** or **yum** update at 04:25 a.m. on the first day of each month, but I commented it out so it no longer runs.

**# 25 04 1 \* \* /usr/bin/dnf -y update**

This line used to perform a monthly update, but I've commented it out.

**Other scheduling tricks**

Now let's do some things that are a little more interesting than these basics. Suppose you want to run a particular job every Thursday at 3 p.m.:

**00 15 \* \* Thu /usr/local/bin/mycronjob.sh**

This line runs**mycronjob.sh**every Thursday at 3 p.m.

Or, maybe you need to run quarterly reports after the end of each quarter. The cron service has no option for "The last day of the month," so instead you can use the first day of the following month, as shown below. (This assumes that the data needed for the reports will be ready when the job is set to run.)

**02 03 1 1,4,7,10 \* /usr/local/bin/reports.sh**

This cron job runs quarterly reports on the first day of the month after a quarter ends.

The following shows a job that runs one minute past every hour between 9:01 a.m. and 5:01 p.m.

**01 09-17 \* \* \* /usr/local/bin/hourlyreminder.sh**

Sometimes you want to run jobs at regular times during normal business hours.

I have encountered situations where I need to run a job every two, three, or four hours. That can be accomplished by dividing the hours by the desired interval, such as **\*/3** for every three hours, or **6-18/3** to run every three hours between 6 a.m. and 6 p.m. Other intervals can be divided similarly; for example, the expression **\*/15** in the minutes position means "run the job every 15 minutes."

**\*/5 08-18/2 \* \* \* /usr/local/bin/mycronjob.sh**

This cron job runs every five minutes during every hour between 8 a.m. and 5:58 p.m.

One thing to note: The division expressions must result in a remainder of zero for the job to run. That's why, in this example, the job is set to run every five minutes (08:05, 08:10, 08:15, etc.) during even-numbered hours from 8 a.m. to 6 p.m., but not during any odd-numbered hours. For example, the job will not run at all from 9 p.m. to 9:59 a.m.

I am sure you can come up with many other possibilities based on these examples.

**Limiting cron access**

Regular users with cron access could make mistakes that, for example, might cause system resources (such as memory and CPU time) to be swamped. To prevent possible misuse, the sysadmin can limit user access by creating a **/etc/cron.allow** file that contains a list of all users with permission to create cron jobs. The root user cannot be prevented from using cron.

By preventing non-root users from creating their own cron jobs, it may be necessary for root to add their cron jobs to the root crontab. "But wait!" you say. "Doesn't that run those jobs as root?" Not necessarily. In the first example in this article, the username field shown in the comments can be used to specify the user ID a job is to have when it runs. This prevents the specified non-root user's jobs from running as root. The following example shows a job definition that runs a job as the user "student":

**04 07 \* \* \* student /usr/local/bin/mycronjob.sh**

If no user is specified, the job is run as the user that owns the crontab file, root in this case.

**cron.d**

The directory **/etc/cron.d** is where some applications, such as [SpamAssassin](http://spamassassin.apache.org/) and [sysstat](https://github.com/sysstat/sysstat), install cron files. Because there is no spamassassin or sysstat user, these programs need a place to locate cron files, so they are placed in **/etc/cron.d**.

The **/etc/cron.d/sysstat** file below contains cron jobs that relate to system activity reporting (SAR). These cron files have the same format as a user cron file.

***# Run system activity accounting tool every 10 minutes*  
\*/10 \* \* \* \* root /usr/lib64/sa/sa1 1 1  
*# Generate a daily summary of process accounting at 23:53*  
53 23 \* \* \* root /usr/lib64/sa/sa2 -A**

The sysstat package installs the**/etc/cron.d/sysstat**cron file to run programs for SAR.

The sysstat cron file has two lines. The first line runs the **sa1** program every 10 minutes to collect data stored in special binary files in the **/var/log/sa** directory. Then, every night at 23:53, the **sa2** program runs to create a daily summary.

**Scheduling tips**

I usually have only a few tasks to schedule on each of my computers, which is simpler than in some of the production and lab environments where I have worked.

One system I administered had around a dozen cron jobs that ran every night and an additional three or four that ran on weekends or the first of the month. That was a challenge, because if too many jobs ran at the same time—especially the backups and compiles—the system would run out of RAM and nearly fill the swap file, which resulted in system thrashing while performance tanked, so nothing got done. We added more memory and improved how we scheduled tasks. We also removed a task that was very poorly written and used large amounts of memory.

The crond service assumes that the host computer runs all the time. That means that if the computer is turned off during a period when cron jobs were scheduled to run, they will not run until the next time they are scheduled. This might cause problems if they are critical cron jobs. Fortunately, there is another option for running jobs at regular intervals: **anacron**.

**anacron**

The [anacron](https://en.wikipedia.org/wiki/Anacron) program performs the same function as crond, but it adds the ability to run jobs that were skipped, such as if the computer was off or otherwise unable to run the job for one or more cycles. This is very useful for laptops and other computers that are turned off or put into sleep mode.

As soon as the computer is turned on and booted, anacron checks to see whether configured jobs missed their last scheduled run. If they have, those jobs run immediately, but only once (no matter how many cycles have been missed). For example, if a weekly job was not run for three weeks because the system was shut down while you were on vacation, it would be run soon after you turn the computer on, but only once, not three times.

The anacron program provides some easy options for running regularly scheduled tasks. Just install your scripts in the **/etc/cron.[hourly|daily|weekly|monthly]** directories, depending how frequently they need to be run.

How does this work? The sequence is simpler than it first appears.

1. The crond service runs the cron job specified in **/etc/cron.d/0hourly**.

***# Run the hourly jobs*  
SHELL=/bin/bash  
PATH=/sbin:/bin:/usr/sbin:/usr/bin  
MAILTO=root  
01 \* \* \* \* root run-parts /etc/cron.hourly**

The contents of**/etc/cron.d/0hourly**cause the shell scripts located in**/etc/cron.hourly**to run.

1. The cron job specified in **/etc/cron.d/0hourly** runs the **run-parts** program once per hour.
2. The **run-parts** program runs all the scripts located in the **/etc/cron.hourly** directory.
3. The **/etc/cron.hourly** directory contains the **0anacron** script, which runs the anacron program using the **/etc/anacrontab** configuration file shown here.

***# /etc/anacrontab: configuration file for anacron*  
  
*# See anacron(8) and anacrontab(5) for details.*  
  
SHELL=/bin/sh  
PATH=/sbin:/bin:/usr/sbin:/usr/bin  
MAILTO=root  
*# the maximal random delay added to the base delay of the jobs*  
RANDOM\_DELAY=45  
*# the jobs will be started during the following hours only*  
START\_HOURS\_RANGE=3-22  
                                                                 
*#period in days   delay in minutes   job-identifier   command*  
1       5       cron.daily              nice run-parts /etc/cron.daily  
7       25      cron.weekly             nice run-parts /etc/cron.weekly  
@monthly 45     cron.monthly            nice run-parts /etc/cron.monthly**

The contents of**/etc/anacrontab**file runs the executable files in the**cron.[daily|weekly|monthly]**directories at the appropriate times.

1. The anacron program runs the programs located in **/etc/cron.daily** once per day; it runs the jobs located in **/etc/cron.weekly** once per week, and the jobs in **cron.monthly** once per month. Note the specified delay times in each line that help prevent these jobs from overlapping themselves and other cron jobs.

Instead of placing complete Bash programs in the **cron.X** directories, I install them in the **/usr/local/bin** directory, which allows me to run them easily from the command line. Then I add a symlink in the appropriate cron directory, such as **/etc/cron.daily**.

The anacron program is not designed to run programs at specific times. Rather, it is intended to run programs at intervals that begin at the specified times, such as 3 a.m. (see the **START\_HOURS\_RANGE** line in the script just above) of each day, on Sunday (to begin the week), and on the first day of the month. If any one or more cycles are missed, anacron will run the missed jobs once, as soon as possible.

**Shortcuts**

The**/etc/anacrontab** file shown above shows us a clue to how we can use shortcuts for a few specific and common times. These single-word time shortcuts can be used to replace the five fields usually used to specify times. The **@** character is used to identify shortcuts to cron. The list below, taken from the crontab(5) man page, shows the shortcuts with their equivalent meanings.

* @reboot : Run once after reboot.
* @yearly : Run once a year, ie. **0 0 1 1 \***
* @annually : Run once a year, ie. **0 0 1 1 \***
* @monthly : Run once a month, ie. **0 0 1 \* \***
* @weekly : Run once a week, ie. **0 0 \* \* 0**
* @daily : Run once a day, ie. **0 0 \* \* \***
* @hourly : Run once an hour, ie. **0 \* \* \* \***

These shortcuts can be used in any of the crontab files, such as those in**/etc/cron.d**.

**More on setting limits**

I use most of these methods for scheduling tasks to run on my computers. All those tasks are ones that need to run with root privileges. It's rare in my experience that regular users really need a cron job. One case was a developer user who needed a cron job to kick off a daily compile in a development lab.

It is important to restrict access to cron functions by non-root users. However, there are circumstances when a user needs to set a task to run at pre-specified times, and cron can allow them to do that. Many users do not understand how to properly configure these tasks using cron and they make mistakes. Those mistakes may be harmless, but, more often than not, they can cause problems. By setting functional policies that cause users to interact with the sysadmin, individual cron jobs are much less likely to interfere with other users and other system functions.

It is possible to set limits on the total resources that can be allocated to individual users or groups, but that is an article for another time.

For more information, the man pages for [cron](http://man7.org/linux/man-pages/man8/cron.8.html" \t "_blank), [crontab](http://man7.org/linux/man-pages/man5/crontab.5.html" \t "_blank), [anacron](http://man7.org/linux/man-pages/man8/anacron.8.html" \t "_blank), [anacrontab](http://man7.org/linux/man-pages/man5/anacrontab.5.html" \t "_blank), and [run-parts](http://manpages.ubuntu.com/manpages/zesty/man8/run-parts.8.html) all have excellent information and descriptions of how the cron system works.