(/) **Wiki**

Cron

From Gentoo Wiki

This article describes how to setup and use cron daemons in Gentoo Linux.

Resources

W Wikipedia (https://en.wikipedia.org/wiki/Cron)

Contents

Cron basics

What cron does

Cron is a daemon that runs scheduled tasks based on input from the command **`crontab`**. It accomplishes this task by waking up every minute and checking to see if there are any cron-jobs to run in any of the user crontabs.

Note
Notice that **`crontab`** is both the name of a list of cron-jobs as well as the name of the command to edit that list.

The de facto cron

There are a few cron implementations to choose from in Portage. All of them offer a similar interface, namely the use of **`crontab`** or a similar command. There is also a related utility called Anacron which is meant to work with cron on systems that are not continuously running.

All of the available cron packages depend on sys-process/cronbase (https://packages.gentoo.org/packages/sys-process/cronbase). This package is not technically depended on by any of the cron packages, but it does provide cron-like functionality that most users can appreciate.

Before getting started working with cron, a proper cron implementation has to be selected.

Which cron is right for the job?

Note
Emerge virtual/cron (https://packages.gentoo.org/packages/virtual/cron) to install Gentoo's default cron implementation.

vixie-cron

Vixie-cron is a full featured cron implementation based on SysV cron. Each user has his own crontab and is allowed to specify environment variables within that crontab. Unlike the other cron variants, it also offers support for SELinux and PAM. It supports fewer architectures than Dcron, but more than Fcron. Latest version is 4.1 released on January 2004.

Features of sys-process/vixie-cron (https://packages.gentoo.org/packages/sys-process/vixie-cron):

- Support for SELinux;
- Support for PAM (`/etc/security/limits.conf`);

- Setting of environment variables in crontabs (PATH, SHELL, HOME, etc.);
- Each user can have a personal crontab; access is controlled by `cron.allow` and `cron.deny`

cronie

Cronie (sys-process/cronie (https://packages.gentoo.org/packages/sys-process/cronie)) is a fork of vixie-cron done by Fedora (https://fedorahosted.org/cronie/wiki). Because of it being a fork it has the same feature set the original vixie-cron provides. Additionally cronie comes with an anacron implementation which must be enabled through the `anacron` USE flag.

dcron (Dillon's Cron)

Dcron (http://www.jimpryor.net/linux/dcron.html) aims to be a simple, elegant and secure implementation of cron. It does not allow the specification of environment variables in crontabs and all cron-jobs are run from `/bin/sh`. Like vixie-cron, each user has his own crontab. As of version 4 it contains anacron-like features.

Features of sys-process/dcron (https://packages.gentoo.org/packages/sys-process/dcron):

- Fast, simple and free of unnecessary features;
- Access to **`crontab`** is limited to the cron group, i.e. it doesn't rely on any external faculties.

fcron

Fcron aims at replacing vixie-cron and anacron. It is designed to work on systems that are not continuously running and it is packed with extra features. It has job startup constraints, job serialization controls, the ability to assign nice values to jobs and the ability to schedule jobs to run at system startup. See fcron's home page (http://fcron.free.fr/) for more information.

Features of sys-process/fcron (https://packages.gentoo.org/packages/sys-process/fcron):

- Designed to work on systems that are not continuously running, i.e. it can run a job after restarting if it was missed;
- Setting of environment variables and many other options in crontabs;
- Enhanced crontab syntax with support for many new features;
- Each user can have a personal crontab, access is controlled by `cron.allow` and `cron.deny`

bcron

Bcron is a new cron system designed with secure operations in mind. To do this, the system is divided into several separate programs, each responsible for a separate task, with strictly controlled communications between them. The user interface is a drop-in replacement for similar systems (such as vixie-cron), but the internals differ greatly. For more information, see the bcron homepage at http://untroubled.org/bcron (http://untroubled.org/bcron).

Features of sys-process/bcron (https://packages.gentoo.org/packages/sys-process/bcron):

- Drop-in replacement for vixie-cron;
- Multiprocess design;

- Native daylight savings time support.

anacron

Anacron is not a cron daemon, it is something that usually works in conjunction with one. It executes commands at intervals specified in days and it does not assume that the system is running continuously; it will run jobs that were missed while the system was down. Anacron usually relies on a cron daemon to run it each day.

Using cron

Installation

Select the right cron implementation for the job, and then emerge it:

```
root # emerge --ask dcron
```

Make sure the cron daemon of choice has been added to the system's init process; without this step the cron daemon will not perform its job.

```
root # /etc/init.d/dcron start
root # rc-update add dcron default
```

Optionally, if Fcron or dcron have **not** been installed, installing Anacron as a helper to the cron daemon might be a wise choice.

```
root # emerge --ask anacron
```

Again, do not forget to add anacron to the system's init process.

```
root # /etc/init.d/anacron start
root # rc-update add anacron default
```

For anacron, there is usually no init process. Instead, anacron needs to be launched through a different cron implementation.

One method is to launch anacron through a cron definition. By default, it installs an hourly run script, which is consumed by most cron implementations by default. If that isn't the case though, then it can still be launched through manual definitions:

FILE `/etc/crontab` **Launching anacron through a cron definition**

```
# Start anacron every 10 minutes
*/10 * * *  root  /usr/sbin/anacron

# Alternatively, run the anacron-provided 0anacron script every hour
# 59 * * * *  root  /etc/cron.hourly/0anacron
```

System crontab

The post install messages from some of these cron packages instruct the user to run **`crontab /etc/crontab`**. The `/etc/crontab` file is the *system crontab*. A cron installation can use it in conjunction with sys-process/cronbase (https://packages.gentoo.org/packages/sys-process/cronbase) to run the scripts in `/etc/cron.{daily,hourly,weekly,monthly}`. Note that only vixie-cron and cronie schedule jobs in `/etc/crontab` automatically. Dcron and fcron users will need to run **`crontab /etc/crontab`** every time they make changes to the `/etc/crontab` file.

Please note that jobs scheduled in the system crontab might not show up in the list of cron-jobs displayed by running **`crontab -l`**.

Of course, users can choose not to use any system crontab at all. If dcron or fcron has been chosen, do **not** run **`crontab /etc/crontab`**. If vixie-cron, cronie or bcron has been chosen comment all lines in `/etc/crontab`.

A quick and easy way to comment out all the lines in a file is by using the sed command. Run the following command to comment out all the lines in `etc/crontab`

```
root # sed -i -e "s/^/#/" /etc/crontab
```

Giving trusted users access to cron

For users other than root to have access to the cron daemon, read this section, otherwise proceed to the next section: Scheduling cron-jobs.

Note
Giving another user access to crontab does not let him run cron-jobs as root. For a user to be able to edit the root crontab, look into using **sudo** (app-admin/sudo (https://packages.gentoo.org/packages/app-admin/sudo)). Please read the Gentoo Sudo(ers) Guide (/wiki/Sudo) for more details.

No matter which cron package has been chosen, to allow a user to use crontab he will first have to be in the cron group. As an example, to add the user *wepy* to the cron group run:

```
root # gpasswd -a wepy cron
```

Note
When adding a user to the cron group, make sure that the user logs out and logs back in for the group change to take effect.

When using **dcron**, the above step is all that is needed to give a user access to crontab. Dcron users may proceed to the next section Scheduling cron-jobs, all others need to keep reading.

When using **fcron**, edit the `/etc/fcron/fcron.deny` and `/etc/fcron/fcron.allow` files. The most secure way to run a system is to first deny all users in `/etc/fcron/fcron.deny`, and then explicitly allow users in `/etc/fcron/fcron.allow`.

Important
If neither `/etc/fcron/fcron.allow` nor `/etc/fcron/fcron.deny` exist then all users in the cron group will be allowed to use crontab. fcron comes with a default `fcron.allow` which **allows all users** in the cron group access to fcrontab.

CODE **Permissions in fcron.deny**

```
all
```

If a user (*wepy* again for this example) should be able to schedule his own cron-jobs, then add him to `/etc/fcron/fcron.allow` as follows:

CODE **Permissions in fcron.allow**

```
wepy
```

If **vixie-cron** or **cronie** has been chosen, then simply edit the `/etc/cron.allow` file.

Important
It is important to note that if only `/etc/cron.allow` exists, then only the cron group users listed there will have access. Otherwise, if only an empty `/etc/cron.deny` exists, then *all* cron group users will be allowed. Do not leave an empty `/etc/cron.deny` if no `/etc/cron.allow` file exists!

For example, to allow access to the user *wepy*, add him to `/etc/cron.allow` as follows:

CODE **Permissions in /etc/cron.allow**

```
wepy
```

Scheduling cron-jobs

The process of editing crontabs is different for each package, but they all support the same basic set of commands: adding and replacing crontabs, editing crontabs, deleting crontabs, and listing cron-jobs in crontabs. The following list shows how to run various commands for each package.

Version	Edit crontab	Remove crontab	New crontab	List cron-jobs
dcron	`crontab -e`	`crontab -d` *`[user]`*	`crontab` *`file`*	`crontab -l`
fcron	`fcrontab -e`	`fcrontab -r` *`[user]`*	`fcrontab` *`file`*	`fcrontab -l`
vixie-cron, cronie & bcron	`crontab -e`	`crontab -r` *`-u [user]`*	`crontab` *`file`*	`crontab -l`

Note
When using the remove command, if no argument is supplied, it deletes the current user's crontab.

Note
Fcron also has a symlink from crontab to fcrontab.

Before any of these commands can be used, first understanding of the crontab itself is needed. Each line in a crontab specifies five time fields in the following order: the minutes (0-59), hours (0-23), days of the month (1-31), months (1-12), and days of the week (0-7, Monday is day 1, Sunday is day 0 and day 7). The days of the week and months can be specified by three-letter abbreviations like mon, tue, jan, feb, etc. Each field can also specify a range of values (e.g. 1-5 or mon-fri), a comma separated list of values (e.g. 1,2,3 or mon,tue,wed) or a range of values with a *step* (e.g. 1-6/2 as 1,3,5).

That sounds a little confusing, but with a few examples it is easy to see it is not as complicated as it sounds.

CODE **Examples**

```
# Run /bin/false every minute year round
*     *     *     *     *        /bin/false

# Run /bin/false at 1:35 on the mon,tue,wed and the 4th of every month
35    1     4     *     mon-wed  /bin/false

# Run /bin/true at 22:25 on the 2nd of March
25    22    2     3     *        /bin/true

# Run /bin/false at 2:00 every Monday, Wednesday and Friday
0     2     *     *     1-5/2    /bin/false
```

Note
Notice how to specify specific days of the week and days of the month before they are combined. If * is used for only one of them, the other takes precedence, while * for both just means every day.

To test what was just covered go through the steps of actually inputting a few cron-jobs. First, create a file called `crons.cron` and make it look like the this:

FILE `crons.cron` **Create a crons.cron file**

```
#Mins  Hours  Days   Months  Day of the week
10     3      1      1       *       /bin/echo "I don't really like cron"
30     16     *      1,2     *       /bin/echo "I like cron a little"
*      *      *      1-12/2  *       /bin/echo "I really like cron"
```

Now add that crontab to the system with the "new command" from the table above.

```
root # crontab crons.cron
```

Note
The output from the echo commands will not be seen unless redirection is used.

To verify the scheduled cron-jobs, use the proper *list command* from the table above.

```
root # crontab -l
```

A list resembling `crons.cron` should be displayed; if not maybe the wrong command was issued to input the crontab.

This crontab should echo "I really like cron" every minute of every hour of every day every other month. Obviously a user would only do that if they really liked cron. The crontab will also echo "I like cron a little" at 16:30 every day in January and February. It will also echo "I don't really like cron" at 3:10 on the January 1st.

If using anacron keep reading this section. Otherwise, proceed to the next section on Editing crontabs.

Anacron users will want to edit `/etc/anacrontab`. This file has four fields: the number of days between each run, the delay in minutes after which it runs, the name of the job, and the command to run.

For example, to have it run **`echo "I like anacron"`** every 5 days, 10 minutes after anacron is started, enter the following:

FILE **`/etc/anacrontab`**

```
5 10 wasting-time /bin/echo "I like anacron"
```

Anacron exits after all of the jobs in anacrontab have finished. To check to see if these jobs should be performed every day, a cron daemon will be used. The instructions at the end of the next section explain how this should be handled.

Editing crontabs

Being realistic, no user would want their system telling them how much they like cron every minute. As a step forward, remove the previous example crontab using the corresponding *remove command* from the table above. Use the corresponding list command to view the cron-jobs afterward to make sure it worked.

```
root # crontab -d
root # crontab -l
```

No cron-jobs should be displayed in the output from **`crontab -l`**. If cron jobs are listed, then the remove command failed to remove the crontab; verify the correct *remove command* for the system's cron package.

Now that we have a clean state, let's put something useful into the **root** crontab. Most people will want to run **`updatedb`** on a weekly basis to make sure that mlocate works properly. To add that to the system's crontab, first edit `crons.cron` again so that it looks like the following:

CODE **A real crontab**

```
22 2 * * 1    /usr/bin/updatedb
```

That would make cron run updatedb at 2:22 A.M. on Monday morning every week. Now input the crontab with the proper *new command* from the table above, and check the list again.

```
root # crontab crons.cron
root # crontab -l
```

Now let's say **emerge --sync** should be ran on a daily schedule in order to keep the Portage tree up to date. This could be done by first editing `crons.cron` and then using **crontab crons.cron** as was done in the example above, *or* by using the proper *edit command* from the table above. This provides a way to edit the user's crontab in situ, without depending on external files like `crons.cron`.

```
root # crontab -e
```

The above command should open the user's crontab with an editor. For example, if **emerge --sync** is to be run every day at 6:30 A.M., make the crontab look something like this:

CODE **A real crontab**

```
22 2 * * 1    /usr/bin/updatedb
30 6 * * *    /usr/bin/emerge --sync
## (if using anacron, add this line)
30 7 * * *    /usr/sbin/anacron -s
```

Again, check the cron-jobs list as done in the previous examples to make sure the jobs are scheduled. If they are all there, then the system is ready to rock and roll.

Using cronbase

As mentioned earlier, all of the available cron packages depend on sys-process/cronbase (https://packages.gentoo.org/packages/sys-process/cronbase). The cronbase package creates `/etc/cron.{hourly,daily,weekly,monthly}`, and a script called `run-crons`. Notice the default `/etc/crontab` file contains something like this:

CODE **Default system crontab**

```
*/15 * * * *     test -x /usr/sbin/run-crons && /usr/sbin/run-crons
0  *  * * *      rm -f /var/spool/cron/lastrun/cron.hourly
0  3  * * *      rm -f /var/spool/cron/lastrun/cron.daily
15 4  * * 6      rm -f /var/spool/cron/lastrun/cron.weekly
30 5  1 * *      rm -f /var/spool/cron/lastrun/cron.monthly
```

To avoid going into much detail, assume these commands will effectively run hourly, daily, weekly and monthly scripts. This method of scheduling cron-jobs has some important advantages:

- They will run even if the computer was off when they were scheduled to run;
- It is easy for package maintainers to place scripts in those well defined places;

- The administrators know exactly where the cron-jobs and crontab are stored, making it easy to backup and restore these parts of their systems.

Note
Again, it is useful to point out that vixie-cron, cronie and bcron automatically read `/etc/crontab`, while dcron and fcron do not. Please read the System crontab section to learn more about this.

Using anacron

As mentioned earlier, anacron is used on systems not meant to be run continuously (like most of the desktop installations). Its default configuration file, `/etc/anacrontab`, is usually similar to the following:

FILE **`/etc/anacrontab`**

```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
# format: period delay job-identifier command
1       5       cron.daily      run-parts /etc/cron.daily
7       10      cron.weekly     run-parts /etc/cron.weekly
30      15      cron.monthly    run-parts /etc/cron.monthly
```

The main difference between this and other common crontabs is that with anacron there is no fixed date/hour for the job scheduling, but only the period between every run. When anacron is started, it will check the contents of a set of files in `/var/spool/anacron` and calculate if the corresponding entry in the configuration file has expired since the last run. If it has, then the command is invoked again.

As a final note, it is important to comment out any overlapping entry in any other cron installed in the system, such as in the following vixie-cron crontab example:

FILE **`/etc/crontab`**

```
# for vixie-cron
# $Header: /var/cvsroot/gentoo-x86/sys-process/vixie-cron/files/crontab-3.0.1-r4,v 1.3 2011/09/20 15:13:51 idl0r Exp $

# Global variables
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# check scripts in cron.hourly, cron.daily, cron.weekly and cron.monthly
59  *  * * *    root    rm -f /var/spool/cron/lastrun/cron.hourly
#9  3  * * *    root    rm -f /var/spool/cron/lastrun/cron.daily
#19 4  * * 6    root    rm -f /var/spool/cron/lastrun/cron.weekly
#29 5  1 * *    root    rm -f /var/spool/cron/lastrun/cron.monthly
*/10  *  * * * root    test -x /usr/sbin/run-crons && /usr/sbin/run-crons @hourly root nice -n 19 run-parts --report /etc/cron.hourl
```

Without doing this, the daily, weekly and monthly parts will be executed - at different times - by both the cron daemon and anacron, leading to possible double job executions.

Troubleshooting

When having problems getting cron to work properly, this quick checklist might be helpful.

Remember, each cron package is different and the range of features varies greatly. Be sure to consult the man pages for crontab, fcrontab, or anacrontab, depending on which cron daemon has been activated!

Is cron running?

To verify that cron is running, see if it shows up in the process list:

```
root # ps ax | grep cron
```

Is cron working?

Try the following:

CODE **crontab to see if cron is running**

```
* * * * * /bin/echo "foobar" >> /file_you_own
```

Then check if `/file_you_own` is modified periodically.

Is the command working?

Same as before, but perhaps redirect the standard error output as well:

CODE **crontab to verify application runs**

```
* * * * * /bin/echo "foobar" >> /file_you_own 2>&1
```

Can cron run the job?

Check the cron log, usually `/var/log/cron.log` or `/var/log/messages` for errors.

Are there any `dead.letters`?

cron usually sends mail when there is a problem; check for mail and look for the creation of a `~/dead.letter` file.

Why are cron mails not sent out?

In order to receive mails from cron, a valid MTA setup must be implemented. This is provided by any package from virtual/mta (https://packages.gentoo.org/packages/virtual/mta).

If the cron mails are only to be sent locally, and not through a fully configured mail server, the system can use mbox (`/var/spool/mail`) mails, by enabling the mbox useflag (https://packages.gentoo.org/useflags/mbox) with the respective package which provides the MTA.

Cron Jobs Alternatives

Some hosting companies do not allow access to cron, but many cron jobs alternatives can be found which are free or commercially available:

- EasyCron (https://www.easycron.com/)

This page is based on a document formerly found on our main website gentoo.org (https://www.gentoo.org/).
The following people contributed to the original document: **Eric Brown, Xavier Neys, Joshua Saddler (nightmorph) (/wiki/User:Nightmorph)**
They are listed here because wiki history does not allow for any external attribution. If you edit the wiki article, please do **not** add yourself here; your contributions are recorded on each article's associated history page.

Retrieved from "http://wiki.gentoo.org/index.php?title=Cron&oldid=695512 (http://wiki.gentoo.org/index.php?title=Cron&oldid=695512)"

Categories (/wiki/Special:Categories): Core system (/wiki/Category:Core_system) | Daemons (/wiki/Category:Daemons)

- This page was last modified on 20 December 2017, at 19:30.