Cron

From **Wikipedia**:

> *cron* is the time-based job scheduler in Unix-like computer operating systems. cron enables users to schedule jobs (commands or shell scripts) to run periodically at certain times or dates. It is commonly used to automate system maintenance or administration.

Related articles

systemd/Timers

Update packages from crontab

Contents

Installation

There are many cron implementations, but none of them are installed by default as the base system uses **systemd/Timers** instead. See the Gentoo's **cron guide (http://www.gentoo.org/doc/en/cron-guide.xml)**, which offers comparisons.

Packages available:

- `cronie (https://www.archlinux.org/packages/?name=cronie)`
- `fcron (https://www.archlinux.org/packages/?name=fcron)`
- `dcron (https://aur.archlinux.org/packages/dcron/)`AUR
- `vixie-cron (https://aur.archlinux.org/packages/vixie-cron/)`AUR
- `scron-git (https://aur.archlinux.org/packages/scron-git/)`AUR

Configuration

Activation and autostart

After installation, the daemon will not be enabled by default. The installed package likely provides a service, which can be controlled by **systemctl**. For example, *cronie* uses `cronie.service`.

Check `/etc/cron.daily/` and similar directories to see which jobs are present. Activating cron service will trigger all of them.

Note: *cronie* provides the `0anacron` *hourly* job, which allows for **delayed runs of other jobs** e.g. if the computer was switched off at the moment of standard execution.

Handling errors of jobs

cron registers the output from *stdout* and *stderr* and attempts to send it as email to the user's spools via the `sendmail` command. Cronie disables mail output if `/usr/bin/sendmail` is not found. In order for mail to be written to a user's spool, there must be an smtp daemon running on the system, e.g. `opensmtpd (https://www.archlinux.org/packages/?name=opensmtpd)`. Otherwise, you can install a package that provides the sendmail command, and configure it to send mail to a remote mail exchanger. You can also log the messages by using the `-m` option and writing a custom script.

Tip: One can send the output to local system users using **Postfix#Local mail**.

1. **Edit** the `cronie.service` unit.
2. Install `esmtp (https://www.archlinux.org/packages/?name=esmtp)`, **msmtp**, `opensmtpd (https://www.archlinux.org/packages/?name=opensmtpd)`, **ssmtp**, or write a custom script.

Example with ssmtp

ssmtp is a send-only sendmail emulator which delivers email from a local computer to an smtp server. While there are currently no active maintainers, it is still by far the simplest way to transfer mail to a configured mailhub. There are no daemons to run, and configuration can be as simple as editing 3 lines in a single configuration file (if your host is trusted to relay unauthenticated email through your mailhub). ssmtp does not receive mail, expand aliases, or manage a queue.

Install `ssmtp` (https://www.archlinux.org/packages/?name=ssmtp), which creates a symbolic link from `/usr/bin/sendmail` to `/usr/bin/ssmtp`. You must then edit `/etc/ssmtp/ssmtp.conf`. See ssmtp for details. Creating a symbolic link to `/usr/bin/sendmail` insures that programs like S-nail (or any package which provides `/usr/bin/mail` will just work without modification.

Restart `cronie` to insure that it detects that you now have a `/usr/bin/sendmail` installed.

Example with msmtp

Install `msmtp-mta` (https://www.archlinux.org/packages/?name=msmtp-mta), which creates a symbolic link from `/usr/bin/sendmail` to `/usr/bin/msmtp`. Restart `cronie` to make sure it detects the new `sendmail` command. You must then provide a way for `msmtp` to convert your username into an email address.

Then either add `MAILTO` line to your crontab, like so:

```
MAILTO=your@email.com
```

or create `/etc/msmtprc` and append this line:

```
aliases /etc/aliases
```

and create `/etc/aliases`:

```
your_username: your@email.com
# Optional:
default: your@email.com
```

Then **modify the configuration** of *cronie* daemon by replacing the `ExecStart` command with:

```
ExecStart=/usr/bin/crond -n -m '/usr/bin/msmtp -t'
```

Example with esmtp

Install **esmtp (https://www.archlinux.org/packages/?name=esmtp)** and **procmail (https://www.archlinux.org/packages/?name=procmail)**.

After installation configure the routing:

```
/etc/esmtprc
-----------------------------------------------
identity myself@myisp.com
       hostname mail.myisp.com:25
       username "myself"
       password "secret"
       starttls enabled
       default
mda "/usr/bin/procmail -d %T"
```

Procmail needs root privileges to work in delivery mode but it is not an issue if you are running the cronjobs as root anyway.

To test that everything works correctly, create a file `message.txt` with `"test message"` in it.

From the same directory run:

```
$ sendmail user_name < message.txt
```

then:

```
$ cat /var/spool/mail/user_name
```

You should now see the test message and the time and date it was sent.

The error output of all jobs will now be redirected to `/var/spool/mail/user_name`.

Due to the privileged issue, it is hard to create and send emails to root (e.g. `su -c ""`). You can ask `esmtp` to forward all root's email to an ordinary user with:

```
/etc/esmtprc
-------------------------------
force_mda="user-name"
```

Note: If the above test didn't work, you may try creating a local configuration in `~/.esmtprc` with the same content.

Run the following command to make sure it has the correct permission:

```
$ chmod 710 ~/.esmtprc
```

Then repeat the test with `message.txt` exactly as before.

Example with opensmtpd

Install **opensmtpd (https://www.archlinux.org/packages/?name=opensmtpd)**.

Edit `/etc/smtpd/smtpd.conf`. The following configuration allows for local delivery:

```
listen on localhost
accept for local deliver to mbox
```

You can proceed to test it. First **start** `smtpd.service`. Then do:

```
$ echo test | sendmail user
```

user can check his/her mail in with any **reader** able to handle mbox format, or just have a look at the file `/var/spool/mail/user`. If everything goes as expected, you can **enable** opensmtpd for future boots.

This approach has the advantage of not sending local cron notifications to a remote server. On the downside, you need a new daemon running.

Note:

- At the moment of writing the Arch opensmtpd package does not create all needed directories under `/var/spool/smtpd`, but the daemon will warn about that specifying the required ownerships and permissions. Just create them as suggested.
- Even though the suggested configuration does not accept remote connections, it's a healthy precaution to add an additional layer of security blocking port 25 with **iptables** or similar.

Long cron job

Suppose this program is invoked by cron :

```
#!/bin/sh
echo "I had a recoverable error!"
sleep 1h
```

What happens is this:

1. cron runs the script

2. as soon as cron sees some output, it runs your MTA, and provides it with the headers. It leaves the pipe open, because the job hasn't finished and there might be more output.
3. the MTA opens the connection to postfix and leaves that connection open while it waits for the rest of the body.
4. postfix closes the idle connection after less than an hour and you get an error like this :

```
smtpmsg='421 … Error: timeout exceeded' errormsg='the server did not accept the mail'
```

To solve this problem you can use the command chronic or sponge from `moreutils (https://www.archlinux.org/packages/?name=moreutils)`. From their respective man page:

chronic
: chronic runs a command, and arranges for its standard out and standard error to only be displayed if the command fails (exits nonzero or crashes). If the command succeeds, any extraneous output will be hidden.

sponge
: sponge reads standard input and writes it out to the specified file. Unlike a shell redirect, sponge soaks up all its input before opening the output file… If no output file is specified, sponge outputs to stdout.

Chronic too buffers the command output before opening its standard output.

Crontab format

The basic format for a crontab is:

```
minute hour day_of_month month day_of_week command
```

- *minute* values can be from 0 to 59.
- *hour* values can be from 0 to 23.
- *day_of_month* values can be from 1 to 31.
- *month* values can be from 1 to 12.
- *day_of_week* values can be from 0 to 6, with 0 denoting Sunday.

Spaces are used to separate fields. To fine-tune your schedule you may also use one of the following symbols:

Symbol	Description
*	Wildcard, specifies every possible time interval
,	List multiple values separated by a comma.
-	Specify a range between two numbers, separated by a hyphen
/	Specify a periodicity/frequency using a slash

For example, the line:

```
*/5 9-16 * 1-5,9-12 1-5 ~/bin/i_love_cron.sh
```

Will execute the script `i_love_cron.sh` at five minute intervals from 9 AM to 4:55 PM on weekdays except during the summer months (June, July, and August). More examples and advanced configuration techniques can be found below.

Besides, crontab has some special keywords:

```
@reboot at startup
@yearly once a year
@annually ( == @yearly)
@monthly once a month
@weekly once a week
@daily once a day
@midnight ( == @daily)
@hourly once an hour
```

For example:

```
@reboot ~/bin/i_love_cron.sh
```

Will execute the script `i_love_cron.sh` at startup.

See more at: **http://www.adminschoice.com/crontab-quick-reference**

Basic commands

Crontabs should never be edited directly; instead, users should use the `crontab` program to work with their crontabs. To be granted access to this command, user must be a member of the users group (see the `gpasswd` command).

To view their crontabs, users should issue the command:

```
$ crontab -l
```

To edit their crontabs, they may use:

```
$ crontab -e
```

Note: By default the `crontab` command uses the `vi` editor. To change it, **export** `EDITOR` or `VISUAL`, or specify the editor directly: `EDITOR=vim crontab -e`.

To remove their crontabs, they should use:

```
$ crontab -r
```

If a user has a saved crontab and would like to completely overwrite their old crontab, he or she should use:

```
$ crontab saved_crontab_filename
```

To overwrite a crontab from the command line (**Wikipedia:stdin**), use

```
$ crontab -
```

To edit somebody else's crontab, issue the following command as root:

```
# crontab -u username -e
```

This same format (appending `-u username` to a command) works for listing and deleting crontabs as well.

Examples

The entry:

```
01 * * * * /bin/echo Hello, world!
```

runs the command `/bin/echo Hello, world!` on the first minute of every hour of every day of every month (i.e. at 12:01, 1:01, 2:01, etc.).

Similarly:

```
*/5 * * jan mon-fri /bin/echo Hello, world!
```

runs the same job every five minutes on weekdays during the month of January (i.e. at 12:00, 12:05, 12:10, etc.).

The line (as noted in "man 5 crontab"):

```
*0,*5 9-16 * 1-5,9-12 1-5 /home/user/bin/i_love_cron.sh
```

will execute the script `i_love_cron.sh` at five minute intervals from 9 AM to 5 PM (excluding 5 PM itself) every weekday (Mon-Fri) of every month except during the summer (June, July, and August).

Periodical settings can also be entered as in this crontab template:

```
# Chronological table of program loadings
# Edit with "crontab" for proper functionality, "man 5 crontab" for formatting
# User: johndoe

# mm  hh  DD  MM  W /path/progam [--option]...  ( W = weekday: 0-6 [Sun=0] )
  21  01  *   *   * /usr/bin/systemctl hibernate
  @weekly           $HOME/.local/bin/trash-empty
```

Here are some self-explanatory crontab syntax examples:

```
30 4 echo "It is now 4:30 am."
0 22 echo "It is now 10 pm."
30 15 25 12 echo "It is 3:30pm on Christmas Day."
30 3 * * * echo "Remind me that it's 3:30am every day."
0 * * * * echo "It is the start of a new hour."
0 6 1,15 * * echo "At 6am on the 1st and 15th of every month."
0 6 * * 2,3,5 echo "At 6am on Tuesday, Wednesday and Thursdays."
59 23 * * 1-5 echo "Just before midnight on weekdays."
0 */2 * * * echo "Every two hours."
```

```
0 20 * * 4 echo "8pm on a Thursday."
0 20 * * Thu echo "8pm on a Thursday."
*/15 9-17 * * 2-5 echo "Every 15 minutes from 9am-5pm on weekdays."
@yearly echo "Happy New Year!"
```

Default editor

To use an alternate default editor, define the `EDITOR` environment variable in a shell initialization script as described in **Environment variables**.

As a regular user, `su` will need to be used instead of `sudo` for the environment variable to be pulled correctly:

```
$ su -c "crontab -e"
```

To have an alias to this `printf` is required to carry the arbitrary string because `su` launches in a new shell:

```
alias scron="su -c $(printf "%q " "crontab -e")"
```

Running X.org server-based applications

Cron does not run under the X.org server therefore it cannot know the environmental variable necessary to be able to start an X.org server application so they will have to be defined. One can use a program like `xuserrun-git (https://aur.archlinux.org/packages/xuserrun-git/)`AUR to do it:

```
17 02 * ... /usr/bin/xuserrun /usr/bin/xclock
```

Or they can be defined manually (`echo $DISPLAY` will give the current DISPLAY value):

```
17 02 * ... env DISPLAY=:0 /usr/bin/xclock
```

If running notify-send for desktop notifications in cron, notify-send is sending values to dbus. So it needs to tell dbus to connect to the right bus. The address can be found by examining DBUS_SESSION_BUS_ADDRESS environment variable and setting it to the same value. Therefore:

```
17 02 * ... env DBUS_SESSION_BUS_ADDRESS=your-address notify-send 'Foo bar'
```

If done through say SSH, permission will need be given:

```
# xhost +si:localuser:$(whoami)
```

Asynchronous job processing

If you regularly turn off your computer but do not want to miss jobs, there are some solutions available (easiest to hardest):

Cronie

`cronie (https://www.archlinux.org/packages/?name=cronie)` comes with anacron included. The project homepage says:

Cronie contains the standard UNIX daemon crond that runs specified programs at scheduled times and related tools. It is based on the original cron and has security and configuration enhancements like the ability to use pam and SELinux.

Dcron

Vanilla `dcron (https://aur.archlinux.org/packages/dcron/)`AUR supports asynchronous job processing. Just put it with @hourly, @daily, @weekly or @monthly with a jobname, like this:

```
@hourly         ID=greatest_ever_job      echo This job is very useful.
```

Cronwhip

`cronwhip (https://aur.archlinux.org/packages/cronwhip/)`AUR is a script to automatically run missed cron jobs; it works with the former default cron implementation, *dcron*. See also the **forum thread (https://bbs.archlinux.org/viewtopic.php?id=57973)**.

Anacron

Anacron is a full replacement for *dcron* which processes jobs asynchronously.

It is provided by `cronie (https://www.archlinux.org/packages/?name=cronie)`. The configuration file is `/etc/anacrontab`. Information on the format can be found in the `anacrontab(5)` **man page**. Running `anacron -T` will test `/etc/anacrontab` for validity.

Fcron

Like *anacron*, `fcron (https://www.archlinux.org/packages/?name=fcron)` assumes the computer is not always running and, unlike *anacron*, it can schedule events at intervals shorter than a single day which may be useful for systems which suspend/hibernate regularly (such as a laptop). Like cronwhip, fcron can run jobs that should have been run during the computer's downtime.

When replacing `cronie (https://www.archlinux.org/packages/?name=cronie)` with fcron be aware the spool directory is `/var/spool/fcron` and the `fcrontab` command is used instead of *crontab* to edit the user crontabs. These crontabs are stored in a binary format with the text version next to them as *foo*.orig in the spool directory. Any scripts which manually edit user crontabs may need to be adjusted due to this difference in behavior.

A quick scriptlet which may aide in converting traditional user crontabs to fcron format:

```
cd /var/spool/cron && (
 for ctab in *; do
  fcrontab ${ctab} -u ${ctab}
 done
)
```

See also the **forum thread (https://bbs.archlinux.org/viewtopic.php?id=140497)**.

Ensuring exclusivity

If you run potentially long-running jobs (e.g., a backup might all of a sudden run for a long time, because of many changes or a particular slow network connection), then `flock` (**`util-linux (https://www.archlinux.org/packages/?name=util-linux)`**) can ensure that the cron job won't start a second time.

```
 5,35 * * * * /usr/bin/flock -n /tmp/lock.backup /root/make-backup.sh
```

Cronie

The relevant file hierarchy for cronie is the following:

```
/etc/
  |----- cron.d/
           | ----- 0hourly
  |----- cron.minutely/
  |----- cron.hourly/
           | ----- 0anacron
  |----- anacrontab
  |----- cron.daily/
  |----- cron.monthly/
  |----- cron.weekly/
  |----- crontab
  |----- cron.deny
```

Cronie provides both *cron* and *anacron* functionalities: *cron* runs jobs at regular time intervals (down to a granularity of one minute) as long as the system is available at the specified time, while *anacron* executes commands at intervals specified in days. Unlike cron, it does not assume that the system is running continuously. Whenever the system is up, *anacron* checks if there are any jobs that should have been run and handles them accordingly.

A *cron* job can be defined in a crontab-like file in the `/etc/cron.d` directory or added within the `/etc/crontab` file. Note the latter is not present by default but is used if it exists. As instructed by `/etc/cron.d/0hourly`, any executable file in `/etc/cron.hourly` will be run every hour (by default at minute 1 of the hour). Files in `/etc/cron.minutely` are executed every minute if instructed accordingly in `/etc/cron.d/0hourly`. The executables are typically shell scripts, symlinks to executable files can also be used. The `/etc/cron.deny` file includes the list of users not allowed to use crontab, without this file, only users listed in `/etc/cron.allow` can use it.

Anacron works similarly, by executing the files in the `/etc/cron.daily`, `/etc/cron.weekly` and `/etc/cron.monthly` directories, placed there depending on the desired job frequency. The cron job `/etc/cron.hourly/0anacron` makes sure that *anacron* is run once daily to perform its pending tasks.

Note: cronie uses `run-parts` to carry out scripts in the different directories. The filenames should not include any dot (.) since `run-parts` in its default mode will silently ignore them (see run-parts(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/run-parts.8)). The names must consist only of upper and lower-case letters, digits, underscores and minus-hyphens.

Note: the output of `systemctl status cronie` might show a message such as `CAN'T OPEN (/etc/crontab): No such file or directory`. However, this can be ignored since cronie does not require one.

Tip: To prevent the sending of output and stop email alert, add `>/dev/null 2>&1` at the end of the line for each cron job to redirect output to /dev/null.

```
0 1 5 10 * /path/to/script.sh >/dev/null 2>&1
```

You can also set `MAILTO=””` variable in your crontab file to disable email alerts.

Dcron

The cron daemon parses a configuration file known as `crontab`. Each user on the system can maintain a separate crontab file to schedule commands individually. The root user's crontab is used to schedule system-wide tasks (though users may opt to use `/etc/crontab` or the `/etc/cron.d` directory, depending on which cron implementation they choose).

```
/var/spool/cron/root
```

```
# Run command at a scheduled time
# Edit this 'crontab -e' for error checking, man 1 crontab for acceptable format

# <@freq>                       <tags and command>
@hourly         ID=sys-hourly   /usr/sbin/run-cron /etc/cron.hourly
@daily          ID=sys-daily    /usr/sbin/run-cron /etc/cron.daily
@weekly         ID=sys-weekly   /usr/sbin/run-cron /etc/cron.weekly
@monthly        ID=sys-monthly  /usr/sbin/run-cron /etc/cron.monthly

# mm  hh  DD  MM  W /path/command (or tags) # W = week: 0-6, Sun=0
  21  01  *   *   * /usr/bin/systemctl suspend
```

These lines exemplify one of the formats that crontab entries can have, namely whitespace-separated fields specifying:

1. @period
2. ID=jobname (this tag is specific to dcron)
3. command

The other standard format for crontab entries is:

1. minute
2. hour

3. day
4. month
5. day of week
6. command

The crontab files themselves are usually stored as `/var/spool/cron/username`. For example, root's crontab is found at `/var/spool/cron/root`

See the crontab **man page** for further information and configuration examples.

See also

- **Gentoo Linux Cron Guide (http://www.gentoo.org/doc/en/cron-guide.xml)**
- **crontab.guru (https://crontab.guru/)** - online editor for cronjob expressions

Retrieved from "https://wiki.archlinux.org/index.php?title=Cron&oldid=500169"