Oracle Linux 9

Automating System Tasks With cron

G24434-02
June 2025

Oracle Linux 9 Automating System Tasks With cron,

G24434-02

Contents

Preface

Oracle Linux 9: Automating System Tasks With cron describes how to automate and schedule system tasks by using `cron`, `anacron`, and Systemd timers.

Documentation License

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also

mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

Automating System Tasks

In Oracle Linux 9, two utilities that are often used for scheduling system tasks, such as performing periodic backups, monitoring the system, and running custom scripts, are `cron` and `anacron`. Both tools automate the running of tasks, also referred to as "jobs," but differ in how those tasks are run. Both utilities automatically run through their respective daemons, so they don't need to be run manually.

Timer unit files for `systemd` can also be used for scheduling tasks. All the utilities described in this document for task automation can be run in combination.

2
Working With cron

Use cron to schedule jobs that run at fixed times, and as often as every minute.

System cron jobs are defined in the cron table in the /etc/crontab configuration file, or in job files stored in the /etc/cron.d directory.

User defined cron jobs are stored in the /var/spool/cron directory and include the user's name in the file name, for example /var/spool/cron/jsmith for the user jsmith.

The crond daemon, which uses the cron utility to run scheduled jobs, checks those locations to decide which jobs need to run.

If the crond daemon identifies a job that was configured to run in the current minute, then the crond daemon runs that job as the owner of the job definition. If the job is a system cron job, then the daemon runs the job as the user that's specified in the job definition, if the user is defined.

If the system is down when a cron job is scheduled to run then, when the system is restarted, the daemon skips that job until the next scheduled run.

cron Table Fields Reference

These lists and examples describe how cron jobs can be configured.

The contents of the /etc/crontab configuration file consist of definitions for the SHELL, PATH, MAILTO, and HOME variables for the environment in which the jobs run. These definitions are then followed by the job definitions themselves. Comment lines start with a # character.

All jobs in the /etc/crontab file are run as the root user, unless otherwise specified.

A /etc/crontab file without any configured job appears as follows:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root

# 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
```

Job definitions consist of information that you specify in the appropriate fields as follows:

Minute
The minute part of the schedule. (0 to 59)

Hour
The hour part of the schedule. (0 to 23)

Day
The calendar day part of the schedule. (1 to 31)

Month
The calendar month part of the schedule. (1 to 12)

Day of the week
The weekday part of the schedule. (0 to 7, or `sun`, `mon`,`tue`, and so on. Sunday is 0 or 7)

Username
User account running the job. For example, `jsmith`. Specifying an asterisk (`*`) runs the job as the owner of the `crontab` file.

Command
The shell script or command to be run. For example, `example.sh`.

For the *minute* through *day-of week* fields, you can use the following special characters:

*****
Specify an asterisk (`*`) to run the `cron` job for all valid intervals in the field.

-
Specify a dash (`-`) to indicate a range of integers. For example, `1-5`.

,
Specify a list of values, separated by commands (`,`). For example, `0,2,4`.

/
Specify a step value by using the slash (`/`). For example, `/3` in the *hour* field. This entry is interpreted as every three hours.

For example, the following entry would run a command every five minutes on weekdays:

```
0-59/5  *  *  *  1-5  * command
```

To run a command at one minute past midnight on the first day of the months April, June, September, and November, add this line:

```
1  0  1  4,6,9,11  *  * command
```

> **Note:**
>
> If you add a job file to the `/etc/cron.hourly` directory, `crond` runs the job every hour.

For more information, see the `crontab(5)` manual page.

Creating a cron Job

Schedule an automated task as a cron job by editing a user's crontab file.

Any user can create a cron job, but the location of the job definition depends on the user's privileges.

An administrator who signs in as root creates jobs that are stored in /etc/crontab. Jobs in /etc/crontab are run as root, unless the job definition specifies a different user.

Any user with administrator privileges can create system-wide cron jobs. The jobs are stored in /etc/crontab.d/ and the job file names include the administrator's username.

A regular user can also create jobs that are stored in /etc/crontab.d/. The job file name also includes that user's name.

To create or edit a crontab file as a signed in user, such as jsmith, follow these steps:

1. Sign in to the system with that user account, for example jsmith.
2. Edit crontab with a text editor by running the following command:

   ```
   crontab -e
   ```

   The crontab command uses the EDITOR environment variable to decide which text editor to use for creating or editing cron jobs in the user's crontab file. To select a specific text editor, such as vim or nano, you can set this environment variable in the shell configuration. To temporarily use a different text editor, run the following command:

   ```
   env EDITOR=text-editor crontab -e
   ```

3. When the editor opens, create a cron job, following the format described in cron Table Fields Reference.

   For example, to define a backup job that runs every 15 minutes and starts a script called mybackup.sh in that user's home directory, add the following line:

   ```
   15 * * * * /home/jsmith/mybackup.sh
   ```

4. Save the file and exit. For the user jsmith, the file is saved as /var/spool/cron/jsmith.

The cron job is now active. To view and verify the contents of the new cron job, run the following command:

```
15 * * * * /home/jsmith/mybackup.sh
```

To delete the signed in user's crontab file, run the following command:

```
crontab -r
```

For more information, see the crontab(5) manual page.

Controlling Access to Running `cron` Jobs

Define which users on the system are allowed to run `cron` jobs by using configuration files to set allowlists and denylists.

The following configuration files manage access control for running `cron` jobs:

- `/etc/cron.allow` contains a list of users that are allowed to run `cron` jobs.
- `/etc/cron.deny` contains a list of users that aren't allowed to run `cron` jobs.

If only the `/etc/cron.deny` configuration file exists, then every user on the system can run `cron` jobs so long as they haven't been listed in that file.

If both configuration files exist, then `/etc/cron.allow` takes precedence and only users that have been listed in that file are allowed to run `cron` jobs. The `/etc/cron.deny` file is ignored in this scenario.

If neither of those configuration files exist, then only the system `root` user can run `cron` jobs.

3
Configuring anacron Jobs

Use anacron to schedule periodic tasks on systems with intermittent rather than continuous uptime.

The anacron utility schedules jobs to be run on a daily, weekly, or monthly interval rather than specifying a particular day or time so that they aren't miss if the system is offline. It was originally intended for use on laptop computers that are routinely suspended or switched off, but it can also be used in an enterprise environment to schedule tasks in persistent cloud instances, containers, and virtual machines that are routinely taken offline to reduce power consumption and hosting costs when they aren't needed.

If anacron isn't already running and the system is connected to mains and not battery power, crond starts anacron automatically.

The crond daemon runs the /etc/cron.hourly/0anacron script as root each hour according to the schedule in /etc/cron.d/0hourly. Then, based on the configuration settings in /etc/anacrontab, the 0anacron script processes the contents in the /etc/cron.daily, /etc/cron.weekly, and /etc/cron.monthly directories.

If a scheduled job hasn't been run because of system downtime, then that job runs when the system restarts.

System anacron jobs are defined in /etc/anacrontab as follows:

```
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 top of the file contains definitions for the SHELL, PATH, MAILTO, RANDOM_DELAY, and START_HOURS_RANGE variables for the environment in which the jobs run, followed by the job definitions themselves. Comment lines start with a # character.

RANDOM_DELAY is the maximum number of random time in minutes that anacron adds to the *delay* parameter for a job. The default minimum delay is 6 minutes. The random offset is intended to prevent anacron overloading the system with too many jobs at the same time.

START_HOURS_RANGE is the time range of hours during the day when anacron can run scheduled jobs.

The bottom part of the file contains job definitions. Each job consists of entries that are spread across 4 columns under the following headings:

period
The frequency of job execution specified in days or as `@daily`, `@weekly`, or `@monthly` for daily, weekly, or monthly.

delay
The number of minutes to wait before running a job.

job-id
The unique name for the job in log files.

command
The shell script or command to be run.

By default, `anacron` runs jobs between 03:00 and 22:00 and delays jobs by between 11 and 50 minutes. The job scripts in `/etc/cron.daily` run between 03:11 and 03:50 every day if the system is running, or after the system is booted and the time is earlier than 22:00. The `run-parts` script sequentially runs every program within the directory specified as its argument.

Scripts in `/etc/cron.weekly` run weekly with a delay offset of between 31 and 70 minutes.

Scripts in `/etc/cron.monthly` run monthly with a delay offset of between 51 and 90 minutes.

For more information, see the `anacron(8)` and `anacrontab(5)` manual pages.

4
Running One-Time Tasks

Use the `at` and `batch` commands for the `atd` service to schedule one-time tasks

Before using these commands, ensure that the `at` service is running:

```
sudo systemctl is-active atd
```

For more information, see the `at(1)` manual page.

To schedule a task to run one time only at a specified tine, use the `at` command.

For example, to schedule a job to run the script at `$HOME/atjob` to run in 20 minutes time, run the following command:

```
at now + 20 minutes < $HOME/atjob
```

```
job 1 at 2025-06-13 11:25
```

To schedule a batch job to run when the system load average is light, use the `batch` command.

For example, to run a batch job to run the script at `$HOME/batchjob` when the system load average is less than 0.8, run the following command:

```
batch < batchjob
```

```
job 2 at 2025-06-13 11:31
```

> **Note:**
>
> The system load average threshold under which you can schedule user-defined batch jobs to run is 0.8, by default. For more information about how that value can vary, see Changing the Behavior of Batch Jobs.

To list all the scheduled one-time jobs that are in queue, run the following command:

```
sudo atq
```

```
job 1 at 2025-06-13 11:25
job 2 at 2025-06-13 11:31
```

To cancel one or more queued jobs, specify their job numbers to the `atrm` command, for example:

```
sudo atrm 2
```

5
Changing the Behavior of Batch Jobs

Change the load-average limit and minimum interval time for batch jobs.

The system load average represents the average number of processes that are queued to run on the CPUs or CPU cores over time. Typically, a system isn't considered overloaded until the load average exceeds 0.8 times the number of CPUs or CPU cores. On such systems, you can use `atd` to run batch jobs when the load average drops to less than the number of CPUs or CPU cores, rather than the default limit of 0.8. For example, on a system with 4 CPU cores, you could set the load-average limit over which `atd` can't run batch jobs to 3.2.

If the batch job often takes more than a minute to run, you can also change the minimum interval that `atd` waits between starting batch jobs. The default minimum interval is 60 seconds.

For more information about monitoring CPU usage and to display the system load average, see Oracle Linux 9: Monitoring and Tuning the System.

1. Open the `/etc/sysconfig/atd` configuration file with a text editor.
2. Uncomment the line that defines the `OPTS` variable.
3. Provide new values for the load average limit and the minimum interval time to the OPTS variable. For example, to set the minimum interval to 100 seconds and the load-average limit to 3, set the following configuration option:

   ```
   OPTS="-b 100 -l 3"
   ```

4. After saving `/etc/sysconfig/atd` configuration file, restart the `atd` service:

   ```
   sudo systemctl restart atd
   ```

5. To verify that the `atd` daemon is running with the new minimum interval and load-average limit, run the following command:

   ```
   sudo systemctl status atd
   ```

   ```
   atd.service - Job spooling tools
     Loaded: loaded (/usr/lib/systemd/system/atd.service; enabled)
     Active: active (running) since Fri 2025-06-13 15:37:04 BST; 2min 53s ago
   Main PID: 6731 (atd)
     CGroup: /system.slice/atd.service
             └─6731 /usr/sbin/atd -f -b 100 -l 3

   Jun 13 15:37:04 localhost.localdomain systemd[1]: Started Job spooling
   tools.
   ```

After these steps have been followed the minimum interval and the load-average limit are changed from the defaults. For more information, see the `systemctl(1)` and `atd(8)` manual pages.

6
Working With Systemd Timers

Use timer unit files in `systemd` to schedule tasks, in a similar way to the `cron` utility that uses `crontab` and other `cron` jobs for the same purpose.

Packages that use specific services to function in the system include their own `systemd` timer unit files. When those packages are installed on Oracle Linux 9, the timer unit files are automatically included. To display active timer unit files, run the following command:

```
systemctl list-unit-files --type=timer
```

> **Note:**
>
> The list of timer files differs depending on where Oracle Linux 9 is running, such as in an Oracle Cloud Infrastructure instance, a physical system, and so on.

Each timer unit file contains parameter settings that manage the schedule of a task. For example, the schedule for running `dnf-makecache.service` is set in the `dnf-makecache.timer` file. To review the contents of that file, run the following command:

```
systemctl cat dnf-makecache.timer
```

```
# /usr/lib/systemd/system/dnf-makecache.timer
[Unit]
Description=dnf makecache --timer
ConditionKernelCommandLine=!rd.live.image
# See comment in dnf-makecache.service
ConditionPathExists=!/run/ostree-booted
Wants=network-online.target

[Timer]
OnBootSec=10min
OnUnitInactiveSec=1h
RandomizedDelaySec=60m
Unit=dnf-makecache.service

[Install]
WantedBy=timers.target
```

The schedule information is specified under the `[Timer]` section. In the sample configuration, the `dnf-makecache.service` service is set to automatically run 10 minutes after the system is booted. The service then goes into idle mode for an hour, as specified by the `OnUnitInactiveSec` parameter. At the end of the hour, the service runs again. This cycle continues every hour indefinitely.

The `RandomizedDelaySec` setting provides a value limit for how much a run can be delayed beyond its schedule. In the example, the service is allowed to run one minute later than its schedule at the latest. This parameter is useful for preventing too many jobs that start at the same time on a specified schedule, which would otherwise risk overloading the resources.

`OnCalendar` is another useful parameter for task scheduling. Suppose that the parameter is set as follows:

```
OnCalendar=*:00/10
```

The `*:00` indicates every hour at the top of the hour, while the `/10` setting indicates 10 minutes. Therefore, the job is set to run hourly, at ten minutes past the top of the hour.

For a complete list of `systemd` timer unit file parameters for scheduling a job, see the `systemd.timer(5)` manual pages.

For more information about using `systemd` with Oracle Linux 9 systems, see Oracle Linux 9: Managing the System With systemd.