

Oracle Linux 9

Monitoring and Tuning the System



F61201-12
February 2025



Oracle Linux 9 Monitoring and Tuning the System,
F61201-12

Copyright © 2022, 2025, Oracle and/or its affiliates.

Contents

Preface

Documentation License	v
Conventions	v
Documentation Accessibility	v
Access to Oracle Support for Accessibility	v
Diversity and Inclusion	v

1 Monitoring the System and Optimizing Performance

System Performance and Monitoring Utilities	1-1
Monitoring the Usage of System Resources	1-2
Monitoring CPU Usage	1-4
Monitoring Memory Usage	1-6
Monitoring Block I/O Usage	1-7
Monitoring File System Usage	1-8
Monitoring Network Usage	1-8
Using the Graphical System Monitor	1-9

2 Working With the sos Command

3 Working With Performance Co-Pilot

4 Working With Tuned

5 Working With the sos Command

6 Working With Performance Co-Pilot

7	Working With TuneD	
<hr/>		
8	Automating System Tasks	
<hr/>		
	Working With cron	8-1
	About the cron Table Fields	8-1
	Creating a cron Job	8-3
	Controlling Access to Running cron Jobs	8-4
	Configuring anacron Jobs	8-4
	Running One-Time Tasks	8-5
	Changing the Behavior of Batch Jobs	8-6
	Working With Systemd Timers	8-7
9	Configuring and Using Auditing	
<hr/>		
	Working With System Log files	9-2
	About Logging Configuration (/etc/rsyslog.conf)	9-3
	Configuring Logwatch	9-6
	Using Process Accounting	9-6
10	Working With Kernel Dumps	
<hr/>		
11	Working With Core Dumps	
<hr/>		
12	Working With the drgn and corelens Kernel Debugging Utilities	
<hr/>		

Preface

[Oracle Linux 9: Monitoring and Tuning the System](#) describes the various utilities, features, and services that you can use to monitor system performance, detect performance issues, and improve the performance of various system components.

Documentation License

The content in this document is licensed under the [Creative Commons Attribution–Share Alike 4.0 \(CC-BY-SA\)](#) license. In accordance with CC-BY-SA, if you distribute this content or an adaptation of it, you must provide attribution to Oracle and retain the original copyright notices.

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.
<i>italic</i>	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

Monitoring the System and Optimizing Performance

Performance issues can be caused by several system components including software or hardware, and any related interactions. Many performance diagnostics utilities are available in Oracle Linux and include tools that monitor and analyze the resource usage of different hardware components, and also tracing tools for diagnosing performance issues in several processes and related threads.

Many performance issues are the result of configuration errors. You can avoid these errors by using a validated configuration that has been pretested for the enabled software, hardware, storage, drivers, and networking components. A validated configuration incorporates best practices for an Oracle Linux deployment and has undergone real-world testing of the complete stack. Oracle publishes many validated configurations, which are available for download. See the release notes for the release that you're running for extra recommendations on kernel parameter settings.

System Performance and Monitoring Utilities

Different Oracle Linux utilities are available that enable you to collect information about system resource usage and errors, and help you to identify performance problems that are caused by overloaded disks, network, memory, or CPUs.

The following packages of the system performance and monitoring utilities are installed by default. If they're not already available, you can install the packages as required.

- `util-linux` provides `dmesg` that displays the contents of the kernel ring buffer, which can contain errors about system resource usage.
- `procps-ng` provides these utilities:
 - `free`: Displays the amount of free and used memory in the system.
 - `top`: Provides a dynamic real-time view of the tasks that are running on a system.
 - `uptime`: Displays the system load averages for the past 1, 5, and 15 minutes.
 - `vmstat`: Reports virtual memory statistics.
- `iproute` provides these utilities:
 - `ip`: Reports network interface statistics and errors.
 - `ss`: Reports network interface statistics.

 **Tip:**

To verify if a utility is available in the system, check if the utility's package is installed. For example, for the `dmesg` utility, you can type:

```
dnf info util-linux

Installed Packages
Name           : util-linux
Version        : version-number
...
```

You can install the following packages to take advantage of extra utilities.

- `sysstat` provides these utilities:
 - `iostat`: Reports I/O statistics.
 - `mpstat`: Reports processor-related statistics.
 - `sar`: Reports information about system activity.
- `iotop` provides `iotop` that monitors disk and swap I/O on a per-process basis.
- `nfs-utils` provides `nfsiostat` that reports I/O statistics for NFS mounts

Many of these utilities provide overlapping functionality. For more information, see the individual manual page for the utility.

For a hands-on tutorial and associated video content on many of these utilities, see [Monitor system resources on Oracle Linux](#).

Monitoring the Usage of System Resources

You can monitor system performance by collecting information about system resources. For better assessment, first, establish a baseline of acceptable measurements under typical operating conditions. You can then use that baseline as a reference point so that you can identify more easily memory shortages, spikes in resource usage, and other problems when they occur. You can also use performance monitoring to plan for future growth and decide how configuration changes might affect future performance.

For more information about monitoring the use of resources in the system, see also [Working With Performance Co-Pilot](#).

Monitoring Usage in Real Time

To run a monitoring command for a set number of seconds in real time and watch the output change, use the `watch` command. For example, you can run the `mpstat` command every second by using the following command:

```
sudo watch -n 1 mpstat
```


That command generates a single-line output that changes information every second, for example:

```
hh:mm:ss CPU    %usr  %nice  %sys %iowait  %irq  %soft  %steal
%guest %gnice %idle
hh:mm:ss all    1.44  0.02   0.80  0.01   0.07  0.05  0.06
0.00    0.00  97.56
```

Alternatively, many of the commands enable you to specify the sampling interval in seconds, for example:

```
sudo mpstat 1
```

The command displays the same information as the previous command, except that the information is in a running list where a new line of information is generated every second, for example:

```
hh:mm:ss CPU    %usr  %nice  %sys %iowait  %irq  %soft  %steal
%guest %gnice %idle
hh:mm:ss all    0.00  0.00   0.00  0.00   0.50  0.50  0.00
0.00    0.00  99.01
hh:mm:ss all    0.00  0.00   0.00  0.00   0.00  0.00  0.00
0.00    0.00 100.00
...
```

To stop real time monitoring, press Ctrl-c.

Monitoring Usage Through Logs

The `sar` utility records statistics every 10 minutes while the system is running and retains that information for every day of the current month. The information is stored in `/var/log/sa/saDD`.

To display the contents of the `sar` log of a specific day of the current month, type:

```
sudo sar -A -f /var/log/sa/saDD
```

You can also use the `sar` command to create a customized log that contains a record of specific information that you want to monitor. Use the following syntax:

```
sudo sar -o datafile seconds count >/dev/null 2>&1 &
```

In the previous command, *datafile* is the full path to the customized log where you want to store the information, while *count* represents the number of samples to record. With this command, the `sar` process runs in the background and collects the data.

To display the results of this monitoring, you would type:

```
sudo sar -A -f datafile
```

Monitoring CPU Usage

When every CPU core is occupied with executing system processes, other processes must wait until a CPU core becomes free or when the scheduler switches a CPU core to run its code. Too many processes that are queued too often can represent a system performance bottleneck.

The following are some commands for monitoring CPU usage. For the different options and arguments that you can use with these commands, see their respective manual pages.

- `uptime`
- `mpstat`
- `sar`
- `top`

The following examples show how you can use these commands to obtain statistics on the system's memory usage:

- Review CPU usage statistics for each CPU core and averaged the data across all the CPU cores.

```
— mpstat -P ALL
```

```
05:10:01 PM CPU   %usr   %nice    %sys    %iowait  %irq   %soft
%steal %guest %gnice   %idle
05:10:01 PM all    0.76   0.02    0.70     0.02   0.08   0.05
0.06   0.00   0.00   98.32
05:10:01 PM  0     0.75   0.01    0.68     0.00   0.09   0.03
0.07   0.00   0.00   98.37
05:10:01 PM  1     0.76   0.03    0.72     0.03   0.06   0.06
0.06   0.00   0.00   98.27
```

```
— sar -u -P ALL
```

```
03:00:01 PM CPU %user  %nice  %system  %iowait  %steal  %idle
03:10:01 PM all  8.30   0.00   2.20     0.22    0.10   89.18
03:10:01 PM  0  8.22   0.00   2.64     0.31    0.09   88.74
03:10:01 PM  1  8.39   0.00   1.77     0.13    0.10   89.61
...
```

The output of these commands include data under the `%idle` heading, which show the percentage of time that a CPU isn't running system or process code. If the percentage is often 0% on all CPU cores, then the system is CPU-bound for the workload that's running.

We recommend that the percentage of time to run system code, which is reported under the `%system` or `%sys` heading, not exceed 30%, especially if `%idle` is close to 0%.

- Review information about system load average.

– uptime

```
21:25:34 up 6:28, 1 user, load average: 0.02, 0.10, 0.04
```

– sar -q

```
14:57:55 LINUX RESTART (2 CPU)

03:00:01 PM runq-sz plist-sz ldavg-1 ldavg-5 ldavg-15 blocked
03:10:01 PM          0      214      0.19      0.30      0.22          0
03:20:01 PM          0      212      0.00      0.05      0.10          0
...
Average:      runq-sz  plist-sz  ldavg-1  ldavg-5  ldavg-15  blocked
Average:          1      212      0.12      0.08      0.05          0
```

The system load average consists of the combination of the number of processes that are running on CPU cores, waiting to run, and waiting for disk I/O activity to complete, which are then averaged over time.

The `uptime` command shows the information in a single line, while the `sar` syntax displays the information in columns under `ldavg-*` headings. Also, the `sar` syntax also shows the number of processes waiting to run as the total number of processes, which are reported under the `runq-sz` and `plist_sz` headings.

On a busy system, we recommend that the load average typically not be greater than two times the number of CPU cores over a period of 5 or 15 minutes. If the load average exceeds four times the number of CPU cores for long periods, then the system is overloaded.

For a better assessment of the system load, find the system's average load under normal loads where users and applications don't experience problems with system responsiveness. Then, look for deviations from this benchmark over time. A dramatic rise in the load average can indicate a serious performance problem.

- Review a real-time listing of CPU activity.

top

```
top - 22:13:34 up 7:16, 1 user, load average: 0.00, 0.02, 0.01
Tasks: 149 total, 1 running, 148 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.2 us, 0.0 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.2 hi, 0.0 si,
0.0 st
MiB Mem : 14705.5 total, 11738.9 free, 753.2 used, 2213.4 buff/cache
MiB Swap: 4096.0 total, 4096.0 free, 0.0 used. 13588.9 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+
COMMAND
    1 root        20   0 118844 18244 11116 S   0.0   0.1   5:18.78
systemd
 34781 root        20   0  11384   7100  1700 S   1.0   0.0   0:00.03
pidstat
  1481 root        20   0 202628 38328 36792 S   0.3   0.3   0:02.94
```

```
sssd_nss  
...
```

By default, `top` lists the most CPU-intensive processes on the system. The output's upper section displays general information including the load averages over the past 1, 5, and 15 minutes, the number of running and sleeping processes or tasks, and total CPU, and memory usage.

The second table displays a list of processes, including the process ID number (PID), the process owner, CPU usage, memory usage, running time, and the command name. By default, the list is sorted by CPU usage, with the top consumer of CPU listed first.

To stop the `top` process, press Ctrl-c.

All the commands can be used together to provide you a picture of the system's CPU usage. For example, sustained large load average or large run queue size and low `%idle` percentages can indicate that the system has insufficient CPU capacity for the workload. When CPU usage is high, the `top` command can identify which processes are likely responsible.

Monitoring Memory Usage

The following are useful utilities to monitor memory usage:

Monitoring Memory Usage With the `sar` Command

To monitor memory usage, use the `sar` command with available options depending on the information you want to obtain. For a list of options, type `sar --help`.

- `sar -r`: Reports memory usage statistics, such as free (`kbmemfree`), available (`kbavail`), and used (`kbmemused`) memory. The report also include `%memused`, which is the percentage of physical memory in use.
- `sar -B`: Reports memory paging statistics, such as page in (`pgpgin/s`) and page out (`pgpgout/s`), page faults and major faults, and so on. The report also includes `pgscank/s`, which is the number of memory pages scanned by the `kswapd` daemon each second, and `pgscand/s`, which is the number of memory pages scanned directly each second.
- `sar -W`: Reports swapping statistics, including `pswpin/s` and `pswpout/s`, which are the numbers of pages each second swapped in and out each second.

If `%memused` is near 100% and the scan rate is continuously over 200 pages each second, the system has a memory shortage.

When a system runs out of real or physical memory and starts using swap space, system performance deteriorates dramatically. If you run out of swap space then some programs or even the entire OS are likely to malfunction. If the `free` or `top` commands indicate that little swap space remains available, then the system is running low on memory.

The output from the `dmesg` command might include notification of any problems with physical memory that were detected at boot time.

Using the Adaptive Memory Management Daemon

To manage memory usage, you can use the Adaptive Memory Management daemon. This daemon is a user space service that monitors free memory on an Oracle Linux system and predicts memory fragmentation and usage. It can also automatically reclaim memory if the system memory becomes too fragmented or is at risk of being filled to capacity.

If the system memory becomes highly fragmented, `adaptivemmd` triggers the kernel to compact memory so that fragmented space can be reclaimed before it's reallocated. If the system is likely to exhaust the available memory then watermarks are adjusted, and this can trigger the kernel to free up new pages in memory. Adaptive Memory Management is available in Unbreakable Enterprise Kernel Release 6 and later.

To use this utility, do the following:

1. Install the `adaptivemm` package.

```
sudo dnf install -y adaptivemm
```

2. Start the daemon service.

```
sudo systemctl enable --now adaptivemmd
```

To see the different options that you can use with the `adaptivemmd` command, type:

```
sudo adaptivemmd -h
```

You can change the configuration options in `/etc/sysconfig/adaptivemmd`.

For more information see the `adaptivemmd(8)` manual page.

Monitoring Block I/O Usage

The `iostat` command monitors the loading of block I/O devices by observing the time that the devices are active relative to the average data transfer rates. You can use this information to adjust the system configuration to balance the I/O loading across disks and host adapters. The following is a sample of the command output:

```
iostat

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.69    0.05   0.77   0.00    0.03   98.46

Device            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda                 1.05         2.32         25.19     611410     6640659
dm-0                 0.62         2.07         20.22     545716     5329660
dm-1                 0.70         0.02         4.97       4632     1308788
```

`iostat -x` reports extended statistics about block I/O activity at one second intervals, including `%util`, which is the percentage of CPU time spent handling I/O requests to a device, and `avgqu-sz`, which is the average queue length of I/O requests that were issued to that device. If `%util` approaches 100% or `avgqu-sz` is greater than 1, device saturation is occurring.

You can also use the `sar -d` command to report on block I/O activity, including values for `%util` and `avgqu-sz`.

The `iotop` utility can help you identify which processes are responsible for excessive disk I/O. `iotop` has a similar user interface to `top`. In its upper section, `iotop` displays the total disk input and output usage in bytes per second. In its lower section, `iotop` displays I/O information for each process, including disk input output usage in bytes per second, the

percentage of time spent swapping in pages from disk or waiting on I/O, and the command name. The following is a sample command output:

```
sudo iotop

Total DISK READ :    0.00 B/s | Total DISK WRITE :        0.00 B/s
Actual DISK READ:    0.00 B/s | Actual DISK WRITE:        0.00 B/s
   TID  PRIO  USER          DISK READ DISK WRITE>
COMMAND
   1 be/4 root          0.00 B/s   0.00 B/s systemd --switched-root --
system --deserialize 16
   2 be/4 root          0.00 B/s   0.00 B/s [kthreadd]
...

```

While you review the output, use the arrow keys to change the sort field, and press **A** to switch the I/O units between bytes each second and total number of bytes, or **O** to switch between displaying all processes or only those processes that are performing I/O.

Monitoring File System Usage

The `sar -v` command reports the number of unused cache entries in the directory cache (`dentunusd`) and the numbers of in-use file handles (`file-nr`), inode handlers (`inode-nr`), and pseudo terminals (`pty-nr`).

```
sar -v

12:00:01 AM dentunusd  file-nr  inode-nr  pty-nr
12:10:33 AM      80101    2944     73074      0
12:20:33 AM      79788    2944     72654      0

```

`nfsiostat` reports I/O statistics for each NFS file system that's mounted. If this command isn't available install the `nfs-utils` package.

Monitoring Network Usage

The `ip -s link` command displays network statistics and errors for all network devices, including the numbers of bytes transmitted (TX) and received (RX). The `dropped` and `overrun` fields provide an indicator of network interface saturation, for example:

```
ip -s link

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    RX: bytes  packets  errors  dropped overrun mcast
        240      4        0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
        240      4        0       0       0       0
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP

```

```
mode DEFAULT group default qlen 1000
  link/ether 08:00:27:60:95:d5 brd ff:ff:ff:ff:ff:ff
  RX: bytes  packets  errors  dropped  overrun mcast
  258187485  671730   0       0         0       17296
  TX: bytes  packets  errors  dropped  carrier collsns
  13227598   130827  0       0         0         0
```

The `ss -s` command displays summary statistics for each protocol, for example:

```
ss -s
```

```
Total: 193
TCP:    9 (estab 2, closed 0, orphaned 0, timewait 0)
```

Using the Graphical System Monitor

The GNOME desktop environment includes a graphical system monitor that you can use to display information about the system configuration, running processes, resource usage, and file systems.

To display the System Monitor, use the following command:

```
gnome-system-monitor
```

Selecting the **Resources** tab displays the following information:

- CPU usage history in graphical form and the current CPU usage as a percentage.
- Memory and swap usage history in graphical form and the current memory and swap usage.
- Network usage history in graphical form, the current network usage for reception and transmission, and the total amount of data received and transmitted.

To display the System Monitor Manual, press `F1` or select Help, then select Contents.

2

Working With the sos Command



Note:

The information in this chapter has been migrated to a separate and more updated documentation. See [Oracle Linux 9: Using sos to Get Debugging Information](#).

3

Working With Performance Co-Pilot

 **Note:**

The information in this chapter has been migrated to a separate and more updated documentation. See [Oracle Linux 9: Collecting and Analyzing Metrics With Performance Co-Pilot](#).

 **Note:**

Performance Co-Pilot (PCP) also provides metrics and other performance-related information that were provided before by the Oracle OSWatcher Black Box (OSWbb) feature, as described in [Oracle Linux 8: Monitoring and Tuning the System](#).

4

Working With TuneD



Note:

The information in this chapter has been migrated to a separate and more updated documentation. See [Oracle Linux 9: Optimize Performance and Power Consumption With TuneD and PowerTop](#).

5

Working With the sos Command



Note:

The information in this chapter has been migrated to a separate and more updated documentation. See [Oracle Linux 9: Using sos to Get Debugging Information](#).

6

Working With Performance Co-Pilot

 **Note:**

The information in this chapter has been migrated to a separate and more updated documentation. See [Oracle Linux 9: Collecting and Analyzing Metrics With Performance Co-Pilot](#).

 **Note:**

Performance Co-Pilot (PCP) also provides metrics and other performance-related information that were provided before by the Oracle OSWatcher Black Box (OSWbb) feature, as described in [Oracle Linux 8: Monitoring and Tuning the System](#).

7

Working With TuneD



Note:

The information in this chapter has been migrated to a separate and more updated documentation. See [Oracle Linux 9: Optimize Performance and Power Consumption With TuneD and PowerTop](#).

8

Automating System Tasks

You can automate tasks to perform periodic backups, monitor the system, run custom scripts, and other administrative tasks. In Oracle Linux, the two utilities that are used for job scheduling are `cron` and `anacron`. Both tools enable you to automate the running of tasks, also referred to as *jobs*, but slightly differ in how the tasks are run. Both utilities automatically run through their respective daemons, and so, you don't need to run these utilities manually.

You can also use `systemd` timer unit files for scheduling tasks. All the utilities described in this document for task automation can work in combination.

Working With `cron`

With `cron`, you can schedule jobs to run as often as every minute. System `cron` jobs are defined in the `cron` table called `/etc/crontab` or in files in the `/etc/cron.d` directory. User defined jobs are stored in the `/var/spool/cron` directory and are named after their users, for example, `/var/spool/cron/jsmith`. The `crond` daemon, which uses the `cron` utility to run scheduled jobs, looks in these locations to decide which jobs need to be run.

If the 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 job is scheduled to run, then when the system is restarted, the daemon bypasses that job until the job's next scheduled run.

For a tutorial on how to work with the `cron` utility, see [Use the Crontab Utility to Schedule Tasks on Oracle Linux](#).

About the `cron` Table Fields

The contents of `/etc/crontab` typically 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.

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

Specify a value of 0-59.

hour

Specify a value of 0-23.

day

Specify a value of 1-31.

month

Specify a value of 1-12 or `jan, feb, ..., dec`.

day-of-week

Specify a value of 0-7 (Sunday is 0 or 7) or `sun, mon, ..., sat`.

user

Specify the user running the command; or, you can specify an asterisk (*), which indicates the owner of the `crontab` file.

command

Specify the shell script or command to be run.

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

Specify an asterisk (*) for all of the valid values for the field.

-

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

,

Specify a list of values, separated by commas (,), 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
```

Run a command at one minute past midnight on the first day of the months April, June, September, and November:

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

**Note:**

If you add an executable job script to the `/etc/cron.hourly` directory, `crond` runs the script every hour.

All jobs in the `/etc/crontab` file are run as `root`.

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

Creating a cron Job

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.
- A user with administrator privileges can create `cron` jobs. The jobs are stored in `/etc/crontab.d/` and named after the administrator's username.
- A regular can create jobs which are stored in `/etc/crontab.d/`. The job file is based after the user's name.

To create or edit a `crontab` file as a user, such as `jsmith`, do the following:

1. Sign in to the system as that user, for example `jsmith`.
2. Edit `crontab` with a text editor.

```
crontab -e
```

If you want to use a specific text editor to create or edit a `cron` job, use the following syntax:

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

3. When the editor opens, create the `cron` job by using the format as described in [About the cron Table Fields](#).

Suppose that you want to define a backup job that you want to run every 15 minutes. Further, you created a script `mybackup.sh` for this task in the user home directory. You would then create the schedule as follows:

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

4. Save the file and exit.
The file is saved as `/var/spool/cron/jsmith`.
5. View and verify the contents of the new `cron` job.

```
crontab -l
```

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


To delete the user `crontab` file, type:

```
crontab -r
```

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

Controlling Access to Running cron Jobs

The following files, each of which contains usernames, manage access control for running cron jobs:

- `/etc/cron.allow` contains the list of users who are permitted to run cron jobs.
- `/etc/cron.deny` contains the list of users who are not permitted to run cron jobs.

If both files exist, then `/etc/cron.allow` takes precedence. Users in this list are permitted to run cron jobs, and `/etc/cron.deny` is ignored. If only `/etc/cron.deny` exists, then only those users that are not on this list can run cron jobs.

If none of these two files exists, then only `root` can run cron jobs.

Configuring anacron Jobs

The `anacron` utility differs with the `cron` utility in that `anacron` limits the number of times a scheduled job can be run to only daily. The `anacron` utility is mainly intended for use on laptop computers.

If `anacron` is not already running and the system is connected to mains and not battery power, `crond` starts `anacron`.

The `crond` daemon runs the `/etc/cron.hourly/0anacron` script as `root` each hour according to the schedule in `/etc/cron.d/0hourly`, which has the following job definition:

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

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

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

delay

Number of minutes to wait before running a job.

job-id

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.

Running One-Time Tasks

Through the `atd` service, you can use certain commands to schedule one-time tasks.

This section describes the use of the `at` and `batch` commands for this purpose. Before you can use these commands, ensure that the `at` service is running.

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

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

Suppose that you have defined a job in `$HOME/atjob`. To schedule that job in 20 minutes time, you would type:

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

```
job 1 at 2021-03-19 11:25
```

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

Suppose that you have defined a batch job in `$HOME/batchjob`. To schedule this job to run if the system load average is less than 0.8, you would type:

```
sudo batch < batchjob
```

```
job 2 at 2013-03-19 11:31
```

 **Note:**

The system load average threshold under which you can schedule user-defined batch jobs to run is 0.8, by default. However, that value can vary. See [Changing the Behavior of Batch Jobs](#).

- To list all the scheduled one-time jobs that are in queue, type:

```
sudo atq
```

```
job 1 at 2021-03-19 11:25  
job 2 at 2013-03-19 11:31
```

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

```
sudo atrm 2
```

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

Changing the Behavior of 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 might not be considered overloaded until the load average exceeds 0.8 times the number of CPUs or CPU cores. On such systems, you might want `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 you know that a batch job typically 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 [Monitoring CPU Usage](#).

To change the load-average limit and minimum interval time for batch jobs:

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:

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

This example sets the minimum interval to 100 seconds and the load-average limit to 3.

4. Restart the `atd` service:

```
sudo systemctl restart atd
```

5. Verify that the `atd` daemon is running with the new minimum interval and load-average limit, for example:

```
sudo systemctl status atd
```

```
atd.service - Job spooling tools
  Loaded: loaded (/usr/lib/systemd/system/atd.service; enabled)
  Active: active (running) since Mon 2014-04-28 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
```

```
Apr 28 15:37:04 localhost.localdomain systemd[1]: Started Job spooling
tools.
```

For more information, see the `systemctl(1)` and `atd(8)` manual pages.

Working With Systemd Timers

Timer unit files are a type of `systemd` file that the `systemctl` utility uses to schedule tasks, similar to the `cron` utility that uses `crontab` and other `cron` jobs for the same purpose.

Typically, packages that use specific services to function in the system include their own `systemd` timer unit files. Thus, when these packages are installed with Oracle Linux, the timer unit files are automatically included. You can display with the timer files in the system with the following command:

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

 **Note:**

The list of timer files might differ depending on where Oracle Linux is running, such as in an instance in Oracle Cloud Infrastructure, 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. The file contains the following settings:

```
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 a tutorial on how to use `systemd` in Oracle Linux, including how to configure `systemd` timer unit files, see [Use systemd on Oracle Linux](#).

9

Configuring and Using Auditing

Auditing collects data at the kernel level that you can then analyze to identify unauthorized activity. Auditing collects data in greater detail than system logging does. The process of examining audit trails to find events of interest can be challenging. Therefore, you might want to consider automating this process.

Some definitions in the audit configuration file, `/etc/audit/auditd.conf`, include the following:

- Data retention policy
- Maximum size of the audit volume
- Action to take if the capacity of the audit volume is exceeded
- Locations of local and remote audit trail volumes

The default audit trail volume is `/var/log/audit/audit.log`. See the `auditd.conf(5)` manual page for more information.

By default, auditing captures specific events such as system logins, modifications to accounts, and `sudo` actions. You can configure auditing to capture detailed system call activity or modifications to certain files. The kernel audit daemon (`auditd`) records the events that you configure, including the event type, a timestamp, the associated user ID, and success or failure of the system call.

The entries in the audit rules file, `/etc/audit/audit.rules`, decides which events are audited. Each rule is a command line option that's passed to the `auditctl` command. Configure this file to match organization's security policy.

The following are examples of rules that you might set in the `/etc/audit/audit.rules` file:

- Record all unsuccessful exits from `open` and `truncate` system calls for files and store the information in the `/etc` directory hierarchy.

```
-a exit,always -S open -S truncate -F /etc -F success=0
```

- Record all files opened by a user with UID 10.

```
-a exit,always -S open -F uid=10
```

- Record all files that have been revised or whose attributes were changed by any user who originally signed in with a UID of 500 or greater.

```
-a exit,always -S open -F auid>=500 -F perm=wa
```

- Record requests for write or for file attribute change access. Store the records in the `/etc/sudoers` file and tag such a record with the string `sudoers-change`.

```
-w /etc/sudoers -p wa -k sudoers-change
```

- Record requests for write and for file attribute change access and store records in the `/etc` directory hierarchy.

```
-w /etc/ -p wa
```

- Require a reboot after changing the audit configuration.

```
-e 2
```

 **Note:**

We recommend that you define rules to reboot at the end of the `/etc/audit/audit.rules` file.

For more examples of audit rules, see also the `auditctl(8)` and `audit.rules(7)` manual pages.

Stringent auditing requirements generate large amounts of audit data and can impose a significant performance overhead. Some site security policies stipulate that a system must shut down if events can't be recorded because the audit volumes have exceeded their capacity. We recommend that you direct audit data to separate file systems in rotation to prevent overspill and to ease backups.

If you tag audit records, then searching an audit volume with the `ausearch` command becomes easier by referring to those tags. For example, to examine records that are tagged with the string `sudoers-change`, you would type:

```
sudo ausearch -k sudoers-change
```

The `aureport` command generates summaries of audit data. For example, the following command generates a report that shows every sign-in event from 1 second after midnight on the previous day until the current time:

```
sudo aureport -l -i -ts yesterday -te now
```

You can set up `cron` jobs or `systemd` timers that run `aureport` periodically to generate reports of interest. See [Use the Crontab Utility to Schedule Tasks on Oracle Linux](#) and [Use systemd on Oracle Linux](#).

See the `ausearch(8)` and `aureport(8)` manual pages for more information.

For a hands-on tutorial on using the auditing tools on Oracle Linux, see [Audit Oracle Linux with Auditd](#).

Working With System Log files

The log files contain messages about the system, kernel, services, and applications. The `journald` logging daemon, which is part of `systemd`, records system messages in nonpersistent journal files in memory. The `journald` daemon forwards messages to the system logging daemon, `rsyslog`.

Files in `/run` are volatile. Thus, the log data is lost after a reboot unless you create the directory `/var/log/journal`. You can use the `journalctl` command to query the journal logs.

For more information, see the `journalctl(1)` and `systemd-journald.service(8)` manual pages.

For a hands-on tutorial introducing system logging tools, see [System Logging on Oracle Linux](#).

About Logging Configuration (`/etc/rsyslog.conf`)

The `/etc/rsyslog.conf` file is the configuration file for the `rsyslogd` daemon. The file contains global directives, module directives, and rules for the daemon or service. By default, `rsyslog` processes and archives only `syslog` messages. However, if required, you can configure `rsyslog` to archive any other messages that `journald` forwards, including kernel, boot, `initrd`, `stdout`, and `stderr` messages.

To obtain the latest information about `rsyslog`, refer to <https://www.rsyslog.com/doc/>. Alternatively, you can install the `rsyslog-doc` package locally.

```
sudo dnf install -y rsyslog-doc
```

The documentation is installed in `/usr/share/doc/rsyslog/html`.

! Important:

The format to configure parameters in `/etc/rsyslog.conf` has changed. The following formats are supported which enables backward compatibility with previous configuration:

- Basic or `sysklogd` format, which has been used since the beginning of system logging.
- Legacy format, where directives are defined on their own specific lines in the file, with each directive being preceded by the dollar (\$) sign, such as `$MainMsgQueueSize`.
- Advanced format, which uses the `RainerScript` scripting language for configuring `rsyslog`.

For more information about these formats, see the relevant sections in <https://www.rsyslog.com/doc/>.

The `/etc/rsyslog.conf` file is divided into the following main sections:

Modules

Modules contain configuration parameters for processing messages. The processed or transformed messages can then be transmitted to various targets as required. Modules are classified into different categories, such as output, input, parser, library, and so on. For a complete list of these module classes, see the appropriate section in <https://www.rsyslog.com/doc/>. For a list of the modules, see the `rsyslog.conf(5)` manual page.

Modules enable different `rsyslog` functionalities to become operative, provided that those modules are loaded. Modules are loaded through the `module load` directive as follows:

```
module(load="module-name")
```

**Note:**

The directive uses the advanced format for loading a module and replaces the `$ModLoad module-name` legacy format.

Global directives

Global directives specify configuration options that apply to the `rsyslogd` daemon. A directive might specify the location of auxiliary files. A directive can also be a `module(load" ")` statement that applies global settings, such as the timestamp format to use for all messages, as shown in the following example:

```
module(load="builtin:omfile" Template=RSYSLOG_TraditionalFileFormat")
```

Because the module applies to all messages, the directive is specified under the Global Directives section.

Rules

Rules or rule sets determine how logged messages are managed.

A rule consists of two fields: a selector field and an action field. The two fields are separated by one more spaces or tabs.

- The selector field has two parts, separated by a period: a facility keyword and a priority keyword. Facility keywords include `auth`, `authpriv`, `cron`, `daemon`, `kern`, and so on. Priority keywords include `debug`, `info`, `notice`, `warning`, and so on. Thus, `kern.*` selects kernel messages of all priority levels, while `kern.emerg` selects emergency kernel messages only.

For a list of both facility and priority selectors, see the `rsyslog.conf(5)` manual page.

- The action field typically indicates to which log file the message content is written. For example, the following rule indicates that `cron` messages are to be logged in `/var/log/cron`:

```
cron.* /var/log/cron
```

You can customize `rsyslog` configuration in two ways:

- Directly change the `/etc/rsyslog.conf` file itself.
- Create a configuration file and store it in `/etc/rsyslog.d`. You might choose this option to prevent custom configurations from being overwritten in case system packages are updated.

Some changes are simple to implement on the `/etc/rsyslog.conf` file, such as changing the log for a specific selector. For example, to change the log for `cron` messages to `cron_new`, you would enter the following:

```
cron.* /var/log/cron_new
```

You then restart the `rsyslog` service for the change to take effect.

Other changes might require more parameter definitions and steps.

Suppose that you want to create a rule that would use TCP to forward messages to another server where the messages are logged. The following steps show you how to implement this sample rule.

1. Create a separate file, for example, `/etc/rsyslog.d/forwarding`, where you would set the parameters for TCP forwarding, similar to the following:

```
*.* action(type="omfwd"
        queue.filename="fwdRule1"
        queue.maxdiskspace="1g"
        queue.saveOnShutdown="on"
        queue.type="linkedlist"
        action.resumeRetryCount="-1"
        target="remote-host.com" port="30514" protocol="tcp"
    )
```

queue.filename

Prefix to be attached to the backup files. The prefixed backup files are created in the location as specified by the `workDir` global directive, for example, `global(workDirectory="/var/log")`.

queue.maxdiskspace

Space limit for log files.

queue.saveOnShutdown

Saves data in memory if `rsyslog` shuts down.

queue.type

Enables a LinkedList in-memory queue.

action.resumeRetryCount

A setting of `-1` means to retry indefinitely if the host is unavailable.

target

Can be a host name or an IP address.

Based on the sample configuration, `rsyslog` forwards messages to the remote server `remote-host.com`. The `rsyslog` service also keeps the message in memory in case the remote server is unavailable. If `rsyslog` shuts down or has exhausted allotted memory, then `rsyslog` creates files on disk with the appropriate prefix to the file names.

2. Open the `/etc/rsyslog.conf` and verify the following:
 - Ensure that the module for TCP syslog reception is loaded. Verify that the comment marks are removed from the following lines:

```
module(load="imtcp")
input(type="imtcp" port="514")
```

- Ensure that the global directive to include `/etc/rsyslog.d` files in `rsyslog` configuration is enabled. Verify that the following line is not commented out:

```
include(file="/etc/rsyslog.d/*.conf" mode="optional")
```

3. Save the file and exit.
4. Restart the `rsyslog` service.

```
systemctl restart rsyslog
```

To manage the rotation and archival of the correct logs, edit `/etc/logrotate.d/syslog` so that it references each of the log files that are defined in the `RULES` section of `/etc/rsyslog.conf`. You can configure how often the logs are rotated and how many past copies of the logs are archived by editing `/etc/logrotate.conf`.

For more information, see the `logrotate(8)`, `logwatch(8)`, `rsyslogd(8)` and `rsyslog.conf(5)` manual pages.

Configuring Logwatch

Logwatch is a monitoring system that you can configure to report on areas of interest in the system logs. After you install the `logwatch` package, the `/etc/cron.daily/0logwatch` script runs every night and sends an email report to `root`. You can set local configuration options in `/etc/logwatch/conf/logwatch.conf` that override the main configuration file `/usr/share/logwatch/default.conf/logwatch.conf`, including the following:

- Log files to monitor, including log files that are stored for other hosts.
- Names of services to monitor, or to be excluded from monitoring.
- Level of detail to report.
- User to be sent an emailed report.

As best practice, configure Logwatch on a log server to monitor the logs for suspicious messages, and disable Logwatch on log clients. However, if you do use Logwatch, disable high precision timestamps by adding the following entry to the `GLOBAL DIRECTIVES` section of `/etc/rsyslog.conf` on each system:

```
module(load="builtin:omfile" Template=RSYSLOG_TraditionalFileFormat")
```

You can also run `logwatch` directly from the command line.

For more information, see the `logwatch(8)` manual page.

Using Process Accounting

The `psacct` package implements the process accounting service in addition to the following utilities that you can use to monitor process activities:

ac

Displays connection times in hours for a user as recorded in the `wtmp` file (by default, `/var/log/wtmp`).

accton

Turns on process accounting to the specified file. If you don't specify a file name argument, process accounting is stopped. The default system accounting file is `/var/account/pacct`.

lastcomm

Displays information about command history as recorded in the system accounting file.

sa

Summarizes information about command history as recorded in the system accounting file.



Note:

As for any logging activity, ensure that the file system has enough space to store the system accounting and `wtmp` files. Monitor the size of the files and truncate them as needed.

For more information, see the `ac(1)`, `accton(8)`, `lastcomm(1)`, and `sa(8)` manual pages.

10

Working With Kernel Dumps



Note:

The information in this chapter has been migrated to a separate and more updated documentation. See [Oracle Linux 9: Managing Kernels and System Boot](#).

11

Working With Core Dumps



Note:

The information in this chapter has been migrated to a separate and more updated documentation. See [Oracle Linux 9: Managing the System With systemd](#).

12

Working With the drgn and corelens Kernel Debugging Utilities

 **Note:**

The information in this chapter has been migrated to a separate and more updated documentation. See [Oracle Linux 9: Debugging the Kernel With Drgn and Corelens](#).