

# Systemd Doku

Doku: [https://fedoraproject.org/wiki/How\\_to\\_debug\\_Systemd\\_problems](https://fedoraproject.org/wiki/How_to_debug_Systemd_problems)  
<https://www.digitalocean.com/community/tutorials/understanding-systemd-units-and-unit-files>

## Systemctl

Alle services listen

```
systemctl list-units --type=service
```

Auf "active" filtern

```
systemctl list-units --type=service --state=active
```

oder Gesamtüberblick über alle, die gerade laufen

```
# systemctl list-units --type=service --state=running  
OR  
# systemctl --type=service --state=running
```

## netstat

Welche Ports werden benutzt?

```
root@radius01:~# netstat -ltup | grep radius  
udp        0      0 0.0.0.0:53441        0.0.0.0:*  
554/freeradius  
udp        0      0 0.0.0.0:18120        0.0.0.0:*  
554/freeradius  
udp        0      0 0.0.0.0:radius       0.0.0.0:*  
554/freeradius  
udp        0      0 0.0.0.0:radius-acct  0.0.0.0:*  
554/freeradius  
udp6       0      0 [::]:radius         [::]:*  
554/freeradius  
udp6       0      0 [::]:radius-acct    [::]:*  
554/freeradius  
udp6       0      0 [::]:42906          [::]:*  
554/freeradius
```

# Journalctl

Siehe <https://www.freedesktop.org/software/systemd/man/systemd-analyze.html>

## Boot Vorgänge auflisten

```
journalctl --list-boots
```

Auflisten aller Boot Vorgänge - 0 ist der neueste

```
journalctl --boot [id]
```

Auswählen eines bestimmten Bootvorgangs

Basic Log Viewing

Auflisten der Logs **vor** dem letzten Bootvorgang

```
'journalctl -b -1'
```

Dasselbe, aber gefiltert auf kernel logs

```
'journalctl -b -1 -k'
```

Mit "GG" wird das Ende des jeweiligen Logs erreicht, also die **letzten Zeilen vor dem Reboot**

To see the logs that the journald daemon has collected, use the journalctl command. When used alone, every journal entry that is in the system will be displayed within a pager (usually less) for you to browse. The oldest entries will be up top:

```
journalctl If you want to display the timestamps in UTC, you can use the -utc flag: journalctl -utc
```

## Journal Filtering by Time

While having access to such a large collection of data is definitely useful, such a large amount of information can be difficult or impossible to inspect and process mentally. Because of this, one of the most important features of journalctl is its filtering options. Displaying Logs from the Current Boot

The most basic of these which you might use daily, is the -b flag. This will show you all of the journal entries that have been collected since the most recent reboot.

```
journalctl -b This will help you identify and manage information that is pertinent to your current environment. In cases where you aren't using this feature and are displaying more than one day of boots, you will see that journalctl has inserted a line that looks like this whenever the system went
```

down: ... - Reboot - ...

This can be used to help you logically separate the information into boot sessions. Past Boots

While you will commonly want to display the information from the current boot, there are certainly times when past boots would be helpful as well. The journal can save information from many previous boots, so journalctl can be made to display information easily.

Some distributions enable saving previous boot information by default, while others disable this feature. To enable persistent boot information, you can either create the directory to store the journal by typing:

```
sudo mkdir -p /var/log/journal Or you can edit the journal configuration
file: sudo nano /etc/systemd/journald.conf
```

Under the [Journal] section, set the Storage= option to "persistent" to enable persistent logging: /etc/systemd/journald.conf

```
... [Journal] Storage=persistent
```

When saving previous boots is enabled on your server, journalctl provides some commands to help you work with boots as a unit of division. To see the boots that journald knows about, use the -list-boots option with journalctl:

```
journalctl -list-boots - 2 caf0524a1d394ce0bdbcff75b94444fe Tue 2015-02-03
21:48:52 UTC-Tue 2015-02-03 22:17:00 UTC - 1 13883d180dc0420db0abcb5fa26d6198
Tue 2015-02-03 22:17:03 UTC-Tue 2015-02-03 22:19:08 UTC 0
bed718b17a73415fade0e4e7f4bea609 Tue 2015-02-03 22:19:12 UTC-Tue 2015-02-03
23:01:01 UTC This will display a line for each boot. The first column is the
offset for the boot that can be used to easily reference the boot with
journalctl. If you need an absolute reference, the boot ID is in the second
column. You can tell the time that the boot session refers to with the two
time specifications listed towards the end. To display information from these
boots, you can use information from either the first or second column. For
instance, to see the journal from the previous boot, use the -l relative
pointer with the -b flag: journalctl -b -l You can also use the boot ID to call
back the data from a boot: journalctl -b caf0524a1d394ce0bdbcff75b94444fe =====
Zeitfenster ===== While seeing log entries by boot is incredibly useful,
often you may wish to request windows of time that do not align well with
system boots. This may be especially true when dealing with long-running
servers with significant uptime. You can filter by arbitrary time limits
using the -since and -until options, which restrict the entries displayed to
those after or before the given time, respectively. The time values can come
in a variety of formats. For absolute time values, you should use the
following format: YYYY-MM-DD HH:MM:SS For instance, we can see all of the
entries since January 10th, 2015 at 5:15 PM by typing: journalctl -since "2015-01-10
17:15:00" If components of the above format are left off, some defaults will be
applied. For instance, if the date is omitted, the current date will be
assumed. If the time component is missing, "00:00:00" (midnight) will be
substituted. The seconds field can be left off as well to default to "00":
journalctl -since "2015-01-10" -until "2015-01-11 03:00" The journal also understands some
relative values and named shortcuts. For instance, you can use the words
```

"yesterday", "today", "tomorrow", or "now". You do relative times by prepending "-" or "+" to a numbered value or using words like "ago" in a sentence construction. To get the data from yesterday, you could type: `journalctl -since yesterday` If you received reports of a service interruption starting at 9:00 AM and continuing until an hour ago, you could type: `journalctl -since 09:00 -until "1 hour ago"` As you can see, it's relatively easy to define flexible windows of time to filter the entries you wish to see. =====

**Filtering by Message Interest =====** We learned above some ways that you can filter the journal data using time constraints. In this section we'll discuss how to filter based on what service or component you are interested in. The systemd journal provides a variety of ways of doing this. By Unit Perhaps the most useful way of filtering is by the unit you are interested in. We can use the `-u` option to filter in this way. For instance, to see all of the logs from an Nginx unit on our system, we can type: `journalctl -u nginx.service` Typically, you would probably want to filter by time as well in order to display the lines you are interested in. For instance, to check on how the service is running today, you can type: `journalctl -u nginx.service -since today` This type of focus becomes extremely helpful when you take advantage of the journal's ability to interleave records from various units. For instance, if your Nginx process is connected to a PHP-FPM unit to process dynamic content, you can merge the entries from both in chronological order by specifying both units: `journalctl -u nginx.service -u php-fpm.service -since today` This can make it much easier to spot the interactions between different programs and debug systems instead of individual processes.

**By Process, User, or Group ID** Some services spawn a variety of child processes to do work. If you have scouted out the exact PID of the process you are interested in, you can filter by that as well. To do this we can filter by specifying the `_PID` field. For instance if the PID we're interested in is 8088, we could type: `journalctl _PID=8088` At other times, you may wish to show all of the entries logged from a specific user or group. This can be done with the `_UID` or `_GID` filters. For instance, if your web server runs under the `www-data` user, you can find the user ID by typing: `id -u www-data` Afterwards, you can use the ID that was returned to filter the journal results: `journalctl _UID=33 -since today` The systemd journal has many fields that can be used for filtering. Some of those are passed from the process being logged and some are applied by `journald` using information it gathers from the system at the time of the log. The leading underscore indicates that the `_PID` field is of the latter type. The journal automatically records and indexes the PID of the process that is logging for later filtering. You can find out about all of the available journal fields by typing: `man systemd.journal-fields` We will be discussing some of these in this guide. For now though, we will go over one more useful option having to do with filtering by these fields. The `-F` option can be used to show all of the available values for a given journal field. For instance, to see which group IDs the systemd journal has entries for, you can type: `journalctl -F _GID` 32 99 102 133 81 This will show you all of the values that the journal has stored for the group ID field. This can help you construct your filters.

**By Component Path** We can also filter by providing a path location. If the path leads to an executable, `journalctl` will display all of the entries that involve the executable in question. For instance, to find those entries that involve the `bash` executable, you can type: `journalctl /usr/bin/bash` Usually, if a unit is available for the executable, that method is cleaner and provides better info (entries

from associated child processes, etc). Sometimes, however, this is not possible. Displaying Kernel Messages Kernel messages, those usually found in dmesg output, can be retrieved from the journal as well. To display only these messages, we can add the `-k` or `-dmesg` flags to our command: `journalctl-k`. By default, this will display the kernel messages from the current boot. You can specify an alternative boot using the normal boot selection flags discussed previously. For instance, to get the messages from five boots ago, you could type: `journalctl-k-b-5` ===== By Priority ===== One filter that system administrators often are interested in is the message priority. While it is often useful to log information at a very verbose level, when actually digesting the available information, low priority logs can be distracting and confusing. You can use `journalctl` to display only messages of a specified priority or above by using the `-p` option. This allows you to filter out lower priority messages. For instance, to show only entries logged at the error level or above, you can type: `journalctl-perr-b`. This will show you all messages marked as error, critical, alert, or emergency. The journal implements the standard syslog message levels. You can use either the priority name or its corresponding numeric value. In order of highest to lowest priority, these are: 0: emerg 1: alert 2: crit 3: err 4: warning 5: notice 6: info 7: debug. The above numbers or names can be used interchangeably with the `-p` option. Selecting a priority will display messages marked at the specified level and those above it.

Modifying the Journal Display Above, we demonstrated entry selection through filtering. There are other ways we can modify the output though. We can adjust the `journalctl` display to fit various needs. Truncate or Expand Output We can adjust how `journalctl` displays data by telling it to shrink or expand the output. By default, `journalctl` will show the entire entry in the pager, allowing the entries to trail off to the right of the screen. This info can be accessed by pressing the right arrow key. If you'd rather have the output truncated, inserting an ellipsis where information has been removed, you can use the `--no-full` option: `journalctl --no-full`. . . Feb 04 20:54:13 journalme sshd[937]: Failed password for root from 83.234.207.60...h2 Feb 04 20:54:13 journalme sshd[937]: Connection closed by 83.234.207.60 [preauth] Feb 04 20:54:13 journalme sshd[937]: PAM 2 more authentication failures; logname...ot You can also go in the opposite direction with this and tell `journalctl` to display all of its information, regardless of whether it includes unprintable characters. We can do this with the `-a` flag: `journalctl-a`.

Output to Standard Out By default, `journalctl` displays output in a pager for easier consumption. If you are planning on processing the data with text manipulation tools, however, you probably want to be able to output to standard output. You can do this with the `--no-pager` option: `journalctl --no-pager`. This can be piped immediately into a processing utility or redirected into a file on disk, depending on your needs.

Output Formats If you are processing journal entries, as mentioned above, you most likely will have an easier time parsing the data if it is in a more consumable format. Luckily, the journal can be displayed in a variety of formats as needed. You can do this using the `-o` option with a format specifier. For instance, you can output the journal entries in JSON by typing: `journalctl -b -u nginx -o json { _ckgedit_ QUOT_CURSOR" : "s=13a21661cf4948289c63075db6c25c00;i=116f1;b=81b58db8fd9046ab9f847ddb82a2fa2d;m=19f0daa;t=50e33c33587ae;x=e307daadb4858635", _ckgedit_ QUOT_REALTIME_TIMESTAMP" : "1422990364739502", _ckgedit_ QUOT_MONOTONIC_TIMESTAMP" : "27200938", _ckgedit_ QUOTBOOT_ID" : "81b58db8fd9046ab9f847ddb82a2fa2d", "PRIORITY" : "6", _ckgedit_ QUOTUID" :`

```

"0", _ckgedit_QUOTGID" : "0", _ckgedit_QUOTCAP_EFFECTIVE" : "3fffffffff",
_ckgedit_QUOTMACHINE_ID" : "752737531a9d1a9c1e3cb52a4ab967ee",
_ckgedit_QUOTHOSTNAME" : "desktop", "SYSLOG_FACILITY" : "3", "CODE_FILE" :
"src/core/unit.c", "CODE_LINE" : "1402", "CODE_FUNCTION" :
"unit_status_log_starting_stopping_reloading", "SYSLOG_IDENTIFIER" :
"systemd", "MESSAGE_ID" : "7d4958e842da4a758f6c1cdc7b36dcc5",
_ckgedit_QUOTTRANSPORT" : "journal", _ckgedit_QUOTPID" : "1",
_ckgedit_QUOTCOMM" : "systemd", _ckgedit_QUOTEXE" :
"/usr/lib/systemd/systemd", _ckgedit_QUOTCMDLINE" :
"/usr/lib/systemd/systemd", _ckgedit_QUOTSYSTEMD_CGROUP" : "/", "UNIT" :
"nginx.service", "MESSAGE" : "Starting A high performance web server and a
reverse proxy server...", _ckgedit_QUOTSOURCE_REALTIME_TIMESTAMP" :
"1422990364737973" } . . . This is useful for parsing with utilities. You
could use the json-pretty format to get a better handle on the data structure
before passing it off to the JSON consumer: journalctl -b -u nginx -o json-
pretty { _ckgedit_QUOT_CURSOR" :
"s=13a21661cf4948289c63075db6c25c00;i=116f1;b=81b58db8fd9046ab9f847ddb82a2fa2d;m=19f0
daa;t=50e33c33587ae;x=e307daadb4858635", _ckgedit_QUOT_REALTIME_TIMESTAMP" :
"1422990364739502", _ckgedit_QUOT_MONOTONIC_TIMESTAMP" : "27200938",
_ckgedit_QUOTBOOT_ID" : "81b58db8fd9046ab9f847ddb82a2fa2d", "PRIORITY" : "6",
_ckgedit_QUOTUID" : "0", _ckgedit_QUOTGID" : "0", _ckgedit_QUOTCAP_EFFECTIVE" : "3fffffffff",
_ckgedit_QUOTMACHINE_ID" : "752737531a9d1a9c1e3cb52a4ab967ee", _ckgedit_QUOTHOSTNAME" :
"desktop", "SYSLOG_FACILITY" : "3", "CODE_FILE" : "src/core/unit.c", "CODE_LINE" : "1402",
"CODE_FUNCTION" : "unit_status_log_starting_stopping_reloading", "SYSLOG_IDENTIFIER" : "systemd",
"MESSAGE_ID" : "7d4958e842da4a758f6c1cdc7b36dcc5", _ckgedit_QUOTTRANSPORT" : "journal",
_ckgedit_QUOTPID" : "1", _ckgedit_QUOTCOMM" : "systemd", _ckgedit_QUOTEXE" :
"/usr/lib/systemd/systemd", _ckgedit_QUOTCMDLINE" : "/usr/lib/systemd/systemd",
_ckgedit_QUOTSYSTEMD_CGROUP" : "/", "UNIT" : "nginx.service", "MESSAGE" : "Starting A high
performance web server and a reverse proxy server...",
_ckgedit_QUOTSOURCE_REALTIME_TIMESTAMP" : "1422990364737973" } . . . The following
formats can be used for display: cat: Displays only the message field itself. export: A
binary format suitable for transferring or backing up. json: Standard JSON with one entry per line.
json-pretty: JSON formatted for better human-readability json-sse: JSON formatted output wrapped to
make add server-sent event compatible short: The default syslog style output short-iso: The default
format augmented to show ISO 8601 wallclock timestamps. short-monotonic: The default format with
monotonic timestamps. short-precise: The default format with microsecond precision verbose: Shows
every journal field available for the entry, including those usually hidden internally. These options
allow you to display the journal entries in the whatever format best suits
your current needs. Active Process Monitoring The journalctl command imitates
how many administrators use tail for monitoring active or recent activity.
This functionality is built into journalctl, allowing you to access these
features without having to pipe to another tool. Displaying Recent Logs To
display a set amount of records, you can use the -n option, which works
exactly as tail -n. By default, it will display the most recent 10 entries:
journalctl -n You can specify the number of entries you'd like to see with a
number after the -n: journalctl -n 20 Following Logs To actively follow the
logs as they are being written, you can use the -f flag. Again, this works as
you might expect if you have experience using tail -f: journalctl -f Journal
Maintenance You may be wondering about the cost is of storing all of the data
we've seen so far. Furthermore, you may be interesting in cleaning up some
older logs and freeing up space. Finding Current Disk Usage You can find out

```

the amount of space that the journal is currently occupying on disk by using the `–disk-usage` flag: `journalctl –disk-usage` Journals take up 8.0M on disk.

**Deleting Old Logs** If you wish to shrink your journal, you can do that in two different ways (available with systemd version 218 and later). If you use the `–vacuum-size` option, you can shrink your journal by indicating a size. This will remove old entries until the total journal space taken up on disk is at the requested size: `sudo journalctl –vacuum-size=1G` Another way that you can shrink the journal is providing a cutoff time with the `–vacuum-time` option. Any entries beyond that time are deleted. This allows you to keep the entries that have been created after a specific time. For instance, to keep entries from the last year, you can type: `sudo journalctl –vacuum-time=1years`

**Limiting Journal Expansion** You can configure your server to place limits on how much space the journal can take up. This can be done by editing the `/etc/systemd/journald.conf` file. The following items can be used to limit the journal growth:

- `SystemMaxUse=`: Specifies the maximum disk space that can be used by the journal in persistent storage.
- `SystemKeepFree=`: Specifies the amount of space that the journal should leave free when adding journal entries to persistent storage.
- `SystemMaxFileSize=`: Controls how large individual journal files can grow to in persistent storage before being rotated.
- `RuntimeMaxUse=`: Specifies the maximum disk space that can be used in volatile storage (within the `/run` filesystem).
- `RuntimeKeepFree=`: Specifies the amount of space to be set aside for other uses when writing data to volatile storage (within the `/run` filesystem).
- `RuntimeMaxFileSize=`: Specifies the amount of space that an individual journal file can take up in volatile storage (within the `/run` filesystem) before being rotated.

By setting these values, you can control how `journald` consumes and preserves space on your server.

**Conclusion** As you can see, the systemd journal is incredibly useful for collecting and managing your system and application data. Most of the flexibility comes from the extensive metadata automatically recorded and the centralized nature of the log. The `journalctl` command makes it easy to take advantage of the advanced features of the journal and to do extensive analysis and relational debugging of different application components.

===== Journalctl Logs bereinigen =====

<https://community.hetzner.com/tutorials/optimize-journalctl-to-save-server-disk-space-in-linux>

===== systemd-analyze =====

**Boot Verlauf prüfen** `systemd-analyze`

**critical-chain** Zeitlicher Verlauf bis zum aktuellen target: `systemd-analyze time`

**Genaue Auflistung aller units:** `systemd-analyze dump`

**Service File prüfen:** `systemd-analyze verify [service file]`

**Grafischer Plot des Startverlaufs für ein Service File** `systemd-analyze dot 'openvpn.*' | dot -Tsvg > openvpn.svg`

===== Anzeigen der Units =====

`<file> systemctl list-unit-files -t target -all </file>` oder `-t timer, -t service`. Ohne `-all` zeigt nur die aktiven: `<code> systemctl list-unit-files -t service </code>`

===== Service > Target Zurdnung =====

`systemctl show -p "Wants" multi-user.target` welche Services werden in einem target angezogen

===== Eigener Runlevel =====

siehe <https://www.mauras.ch/systemd-run-it-last.html>, umgesetzt auf tarapiroe für vbox. Es wird ein eigenes target in `/etc/systemd/system` angelegt. Dieses target wird **nach** multi-user gestartet.

**AllowIsolate=** Takes a boolean argument. If true, this unit may be used with the **systemctl isolate** command. Otherwise, this will be refused. It probably is a good idea to leave this disabled except for target units that shall be used similar to runlevels in SysV init systems, just as a precaution to avoid unusable system states. This option defaults to false.

[Unit] Description = Custom Target Requires = multi-user.target After = multi-user.target AllowIsolate = yes

Die units für dieses Target sehen so aus: [Unit] Description = My last command After = multi-user.target

[Service] Type = simple ExecStart = /sbin/last\_command [Install] WantedBy = custom.target

Dann

eigenes Verzeichnis `/etc/systemd/system/custom.target.wants` anlegen, darin symlinks auf die units, die in diesem Level gestartet werden sollen. Danach `systemctl daemon-reload`, um Änderungen einzulesen. Dieser Befehl zeigt alle targets inkl. der nicht laufenden: `systemctl list-units -type target -all` Dies schaltet auf das neue target um. `systemctl isolate custom.target` Damit wird das neue Target als default gesetzt (`show-default` zeigt das aktuelle default target):  
`systemctl set-default multi-user.target` ===== Starten von Services auf Systemen mit NetworkManager ===== siehe

<https://unix.stackexchange.com/questions/126009/cause-a-script-to-execute-after-networking-has-started/126146#126146> It is very easy to affect systemd's unit ordering. On the other hand you need to be careful about what a completed unit guarantees. ===== Configure your service ===== On current systems, ordering after `network.target` just guarantees that the network service has been started, not that there's some actual configuration. You need to order after `network-online.target` and pull it in to achieve that.

[Unit] Wants=`network-online.target` After=`network-online.target` For compatibility with older systems, you may need to order after `network.target` as well.

[Unit] Wants=`network-online.target` After=`network.target` `network-online.target` That's for the unit file of your service and for systemd.

From:

<https://wiki.netzwissen.de/> - **netzwissen.de Wiki**

Permanent link:

[https://wiki.netzwissen.de/doku.php?id=systemd\\_journalctl&rev=1748884719](https://wiki.netzwissen.de/doku.php?id=systemd_journalctl&rev=1748884719)

Last update: **02/06/2025 - 17:18**

