NAME

systemd-analyze − Analyze and debug system manager

SYNOPSIS

systemd−analyze [OPTIONS...] [time]

systemd−analyze [OPTIONS...] blame

systemd−analyze [OPTIONS...] critical−chain [*UNIT*...]

systemd−analyze [OPTIONS...] dump

systemd−analyze [OPTIONS...] plot [>file.svg]

systemd−analyze [OPTIONS...] dot [*PATTERN*...] [>file.dot]

systemd−analyze [OPTIONS...] unit−paths

systemd−analyze [OPTIONS...] exit−status [*STATUS*...]

systemd−analyze [OPTIONS...] capability [*CAPABILITY*...]

systemd−analyze [OPTIONS...] condition *CONDITION*...

systemd−analyze [OPTIONS...] syscall−filter [*SET*...]

systemd−analyze [OPTIONS...] calendar *SPEC*...

systemd−analyze [OPTIONS...] timestamp *TIMESTAMP*...

systemd−analyze [OPTIONS...] timespan *SPAN*...

systemd−analyze [OPTIONS...] cat−config *NAME*|*PATH*...

systemd−analyze [OPTIONS...] verify [*FILE*...]

systemd−analyze [OPTIONS...] security *UNIT*...

DESCRIPTION

systemd−analyze may be used to determine system boot−up performance statistics and retrieve other state and tracing information from the system and service manager, and to verify the correctness of unit files. It is also used to access special functions useful for advanced system manager debugging.

If no command is passed, **systemd−analyze time** is implied.

systemd−analyze time

This command prints the time spent in the kernel before userspace has been reached, the time spent in the initial RAM disk (initrd) before normal system userspace has been reached, and the time normal system userspace took to initialize. Note that these measurements simply measure the time passed up to the point where all system services have been spawned, but not necessarily until they fully finished initialization or the disk is idle.

Example 1. Show how long the boot took

```
# in a container
$ systemd−analyze time
Startup finished in 296ms (userspace)
multi−user.target reached after 275ms in userspace

# on a real machine
$ systemd−analyze time
Startup finished in 2.584s (kernel) + 19.176s (initrd) + 47.847s (userspace) = 1min 9.608s
multi−user.target reached after 47.820s in userspace
```

systemd−analyze blame

This command prints a list of all running units, ordered by the time they took to initialize. This information may be used to optimize boot−up times. Note that the output might be misleading as the initialization of one service might be slow simply because it waits for the initialization of another service to complete. Also note: **systemd−analyze blame** doesn't display results for services with *Type=simple*, because systemd considers such services to be started immediately, hence no measurement of the initialization delays can be done. Also note that this command only shows the time units took for starting up, it does not show how long unit jobs spent in the execution queue. In particular it shows the time units spent in "activating" state, which is not defined for units such as device units that transition directly from "inactive" to "active". This command hence gives an impression of the performance of program code, but cannot accurately reflect latency introduced by waiting for hardware and similar events.

Example 2. Show which units took the most time during boot

```
$ systemd−analyze blame
         32.875s pmlogger.service
         20.905s systemd−networkd−wait−online.service
         13.299s dev−vda1.device
         ...
            23ms sysroot.mount
            11ms initrd−udevadm−cleanup−db.service
             3ms sys−kernel−config.mount
```

systemd−analyze critical−chain [*UNIT*...]

This command prints a tree of the time−critical chain of units (for each of the specified *UNIT*s or for the default target otherwise). The time after the unit is active or started is printed after the "@" character. The time the unit takes to start is printed after the "+" character. Note that the output might be misleading as the initialization of services might depend on socket activation and because of the parallel execution of units. Also, similar to the **blame** command, this only takes into account the time units spent in "activating" state, and hence does not cover units that never went through an "activating" state (such as device units that transition directly from "inactive" to "active"). Moreover it does not show information on jobs (and in particular not jobs that timed out).

Example 3. systemd−analyze critical−chain

```
$ systemd−analyze critical−chain
multi−user.target @47.820s
pmie.service @35.968s +548ms
  pmcd.service @33.715s +2.247s
    network−online.target @33.712s
      systemd−networkd−wait−online.service @12.804s +20.905s
        systemd−networkd.service @11.109s +1.690s
          systemd−udevd.service @9.201s +1.904s
            systemd−tmpfiles−setup−dev.service @7.306s +1.776s
              kmod−static−nodes.service @6.976s +177ms
                systemd−journald.socket
                  system.slice
                    −.slice
```

systemd−analyze dump

This command outputs a (usually very long) human−readable serialization of the complete server state. Its format is subject to change without notice and should not be parsed by applications.

Example 4. Show the internal state of user manager

```
$ systemd−analyze −−user dump
Timestamp userspace: Thu 2019−03−14 23:28:07 CET
Timestamp finish: Thu 2019−03−14 23:28:07 CET
Timestamp generators−start: Thu 2019−03−14 23:28:07 CET
Timestamp generators−finish: Thu 2019−03−14 23:28:07 CET
```

```
Timestamp units−load−start: Thu 2019−03−14 23:28:07 CET
Timestamp units−load−finish: Thu 2019−03−14 23:28:07 CET
−> Unit proc−timer_list.mount:
      Description: /proc/timer_list
      ...
−> Unit default.target:
      Description: Main user target
...
```

systemd−analyze plot

This command prints an SVG graphic detailing which system services have been started at what time, highlighting the time they spent on initialization.

Example 5. Plot a bootchart

```
$ systemd−analyze plot >bootup.svg
$ eog bootup.svg&
```

systemd−analyze dot [*pattern*...]

This command generates textual dependency graph description in dot format for further processing with the GraphViz **dot**(1) tool. Use a command line like **systemd−analyze dot | dot −Tsvg >systemd.svg** to generate a graphical dependency tree. Unless **−−order** or **−−require** is passed, the generated graph will show both ordering and requirement dependencies. Optional pattern globbing style specifications (e.g. *.target) may be given at the end. A unit dependency is included in the graph if any of these patterns match either the origin or destination node.

Example 6. Plot all dependencies of any unit whose name starts with "avahi−daemon"

```
$ systemd−analyze dot 'avahi−daemon.*' | dot −Tsvg >avahi.svg
$ eog avahi.svg
```

Example 7. Plot the dependencies between all known target units

```
$ systemd−analyze dot −−to−pattern='*.target' −−from−pattern='*.target' \
     | dot −Tsvg >targets.svg
$ eog targets.svg
```

systemd−analyze unit−paths

This command outputs a list of all directories from which unit files, .d overrides, and .wants, .requires symlinks may be loaded. Combine with **−−user** to retrieve the list for the user manager instance, and **−−global** for the global configuration of user manager instances.

Example 8. Show all paths for generated units

```
$ systemd−analyze unit−paths | grep '^/run'
/run/systemd/system.control
/run/systemd/transient
/run/systemd/generator.early
/run/systemd/system
/run/systemd/system.attached
/run/systemd/generator
/run/systemd/generator.late
```

Note that this verb prints the list that is compiled into **systemd−analyze** itself, and does not communicate with the running manager. Use

```
systemctl [−−user] [−−global] show −p UnitPath −−value
```

to retrieve the actual list that the manager uses, with any empty directories omitted.

systemd−analyze exit−status [*STATUS*...]

This command prints a list of exit statuses along with their "class", i.e. the source of the definition (one of "glibc", "systemd", "LSB", or "BSD"), see the Process Exit Codes section in **systemd.exec**(5). If no additional arguments are specified, all known statuses are shown. Otherwise, only the definitions for the specified codes are shown.

Example 9. Show some example exit status names

```
$ systemd−analyze exit−status 0 1 {63..65}
NAME    STATUS CLASS
SUCCESS 0      glibc
FAILURE 1      glibc
−       63     −
USAGE   64     BSD
DATAERR 65     BSD
```

systemd−analyze capability [*CAPABILITY*...]

This command prints a list of Linux capabilities along with their numeric IDs. See **capabilities**(7) for details. If no argument is specified the full list of capabilities known to the service manager and the kernel is shown. Capabilities defined by the kernel but not known to the service manager are shown as "cap_???". Optionally, if arguments are specified they may refer to specific cabilities by name or numeric ID, in which case only the indicated capabilities are shown in the table.

Example 10. Show some example capability names

```
$ systemd−analyze capability 0 1 {30..32}
NAME              NUMBER
cap_chown              0
cap_dac_override       1
cap_audit_control     30
cap_setfcap           31
cap_mac_override      32
```

systemd−analyze condition *CONDITION*...

This command will evaluate *Condition*=... and *Assert*=... assignments, and print their values, and the resulting value of the combined condition set. See **systemd.unit**(5) for a list of available conditions and asserts.

Example 11. Evaluate conditions that check kernel versions

```
$ systemd−analyze condition 'ConditionKernelVersion = ! <4.0' \
        'ConditionKernelVersion = >=5.1' \
        'ConditionACPower=|false' \
        'ConditionArchitecture=|!arm' \
        'AssertPathExists=/etc/os−release'
test.service: AssertPathExists=/etc/os−release succeeded.
Asserts succeeded.
test.service: ConditionArchitecture=|!arm succeeded.
test.service: ConditionACPower=|false failed.
test.service: ConditionKernelVersion=>=5.1 succeeded.
test.service: ConditionKernelVersion=!<4.0 succeeded.
Conditions succeeded.
```

systemd−analyze syscall−filter [*SET*...]

This command will list system calls contained in the specified system call set *SET*, or all known sets if no sets are specified. Argument *SET* must include the "@" prefix.

systemd−analyze calendar *EXPRESSION*...

This command will parse and normalize repetitive calendar time events, and will calculate when they elapse next. This takes the same input as the *OnCalendar=* setting in **systemd.timer**(5), following the syntax described in **systemd.time**(7). By default, only the next time the calendar expression will elapse is shown; use **−−iterations=** to show the specified number of next times the expression elapses. Each time the expression elapses forms a timestamp, see the **timestamp** verb below.

Example 12. Show leap days in the near future

```
$ systemd−analyze calendar −−iterations=5 '*−2−29 0:0:0'
  Original form: *−2−29 0:0:0
Normalized form: *−02−29 00:00:00
    Next elapse: Sat 2020−02−29 00:00:00 UTC
       From now: 11 months 15 days left
       Iter. #2: Thu 2024−02−29 00:00:00 UTC
       From now: 4 years 11 months left
       Iter. #3: Tue 2028−02−29 00:00:00 UTC
       From now: 8 years 11 months left
       Iter. #4: Sun 2032−02−29 00:00:00 UTC
       From now: 12 years 11 months left
       Iter. #5: Fri 2036−02−29 00:00:00 UTC
       From now: 16 years 11 months left
```

systemd−analyze timestamp *TIMESTAMP*...

This command parses a timestamp (i.e. a single point in time) and outputs the normalized form and the difference between this timestamp and now. The timestamp should adhere to the syntax documented in **systemd.time**(7), section "PARSING TIMESTAMPS".

Example 13. Show parsing of timestamps

```
$ systemd−analyze timestamp yesterday now tomorrow
  Original form: yesterday
Normalized form: Mon 2019−05−20 00:00:00 CEST
       (in UTC): Sun 2019−05−19 22:00:00 UTC
   UNIX seconds: @15583032000
       From now: 1 day 9h ago

  Original form: now
Normalized form: Tue 2019−05−21 09:48:39 CEST
       (in UTC): Tue 2019−05−21 07:48:39 UTC
   UNIX seconds: @1558424919.659757
       From now: 43us ago

  Original form: tomorrow
Normalized form: Wed 2019−05−22 00:00:00 CEST
       (in UTC): Tue 2019−05−21 22:00:00 UTC
   UNIX seconds: @15584760000
       From now: 14h left
```

systemd−analyze timespan *EXPRESSION*...

This command parses a time span (i.e. a difference between two timestamps) and outputs the normalized form and the equivalent value in microseconds. The time span should adhere to the syntax documented in **systemd.time**(7), section "PARSING TIME SPANS". Values without units are parsed as seconds.

Example 14. Show parsing of timespans

```
$ systemd−analyze timespan 1s 300s '1year 0.000001s'
```

```
Original: 1s
      μs: 1000000
   Human: 1s

Original: 300s
      μs: 300000000
   Human: 5min

Original: 1year 0.000001s
      μs: 31557600000001
   Human: 1y 1us
```

systemd−analyze cat−config *NAME|PATH*...

This command is similar to **systemctl cat**, but operates on config files. It will copy the contents of a config file and any drop−ins to standard output, using the usual systemd set of directories and rules for precedence. Each argument must be either an absolute path including the prefix (such as /etc/systemd/logind.conf or /usr/lib/systemd/logind.conf), or a name relative to the prefix (such as systemd/logind.conf).

Example 15. Showing logind configuration

```
$ systemd−analyze cat−config systemd/logind.conf
# /etc/systemd/logind.conf
...
[Login]
NAutoVTs=8
...

# /usr/lib/systemd/logind.conf.d/20−test.conf
... some override from another package

# /etc/systemd/logind.conf.d/50−override.conf
... some administrator override
```

systemd−analyze verify *FILE*...

This command will load unit files and print warnings if any errors are detected. Files specified on the command line will be loaded, but also any other units referenced by them. The full unit search path is formed by combining the directories for all command line arguments, and the usual unit load paths. The variable *$SYSTEMD_UNIT_PATH* is supported, and may be used to replace or augment the compiled in set of unit load paths; see **systemd.unit**(5). All units files present in the directories containing the command line arguments will be used in preference to the other paths.

The following errors are currently detected:

- unknown sections and directives,
- missing dependencies which are required to start the given unit,
- man pages listed in *Documentation=* which are not found in the system,
- commands listed in *ExecStart=* and similar which are not found in the system or not executable.

Example 16. Misspelt directives

```
$ cat ./user.slice
[Unit]
WhatIsThis=11
Documentation=man:nosuchfile(1)
Requires=different.service
```

```
[Service]
Description=x

$ systemd−analyze verify ./user.slice
[./user.slice:9] Unknown lvalue 'WhatIsThis' in section 'Unit'
[./user.slice:13] Unknown section 'Service'. Ignoring.
Error: org.freedesktop.systemd1.LoadFailed:
   Unit different.service failed to load:
   No such file or directory.
Failed to create user.slice/start: Invalid argument
user.slice: man nosuchfile(1) command failed with code 16
```

Example 17. Missing service units

```
$ tail ./a.socket ./b.socket
==> ./a.socket <==
[Socket]
ListenStream=100

==> ./b.socket <==
[Socket]
ListenStream=100
Accept=yes

$ systemd−analyze verify ./a.socket ./b.socket
Service a.service not loaded, a.socket cannot be started.
Service b@0.service not loaded, b.socket cannot be started.
```

systemd−analyze security [*UNIT*...]

This command analyzes the security and sandboxing settings of one or more specified service units. If at least one unit name is specified the security settings of the specified service units are inspected and a detailed analysis is shown. If no unit name is specified, all currently loaded, long−running service units are inspected and a terse table with results shown. The command checks for various security−related service settings, assigning each a numeric "exposure level" value, depending on how important a setting is. It then calculates an overall exposure level for the whole unit, which is an estimation in the range 0.0...10.0 indicating how exposed a service is security−wise. High exposure levels indicate very little applied sandboxing. Low exposure levels indicate tight sandboxing and strongest security restrictions. Note that this only analyzes the per−service security features systemd itself implements. This means that any additional security mechanisms applied by the service code itself are not accounted for. The exposure level determined this way should not be misunderstood: a high exposure level neither means that there is no effective sandboxing applied by the service code itself, nor that the service is actually vulnerable to remote or local attacks. High exposure levels do indicate however that most likely the service might benefit from additional settings applied to them.

Please note that many of the security and sandboxing settings individually can be circumvented — unless combined with others. For example, if a service retains the privilege to establish or undo mount points many of the sandboxing options can be undone by the service code itself. Due to that is essential that each service uses the most comprehensive and strict sandboxing and security settings possible. The tool will take into account some of these combinations and relationships between the settings, but not all. Also note that the security and sandboxing settings analyzed here only apply to the operations executed by the service code itself. If a service has access to an IPC system (such as D−Bus) it might request operations from other services that are not subject to the same restrictions. Any comprehensive security and sandboxing analysis is hence incomplete if the IPC access policy is not validated too.

Example 18. Analyze systemd−logind.service

```
$ systemd−analyze security −−no−pager systemd−logind.service
  NAME                DESCRIPTION                              EXPOSURE
 PrivateNetwork=     Service has access to the host's network      0.5
 User=/DynamicUser=  Service runs as root user                     0.4
 DeviceAllow=        Service has no device ACL                     0.2
✓ IPAddressDeny=      Service blocks all IP address ranges
...
→ Overall exposure level for systemd−logind.service: 4.1 OK
```

OPTIONS

The following options are understood:

−−system
Operates on the system systemd instance. This is the implied default.

−−user
Operates on the user systemd instance.

−−global
Operates on the system−wide configuration for user systemd instance.

−−order, **−−require**
When used in conjunction with the **dot** command (see above), selects which dependencies are shown in the dependency graph. If **−−order** is passed, only dependencies of type *After*= or *Before*= are shown. If **−−require** is passed, only dependencies of type *Requires*=, *Requisite*=, *Wants*= and *Conflicts*= are shown. If neither is passed, this shows dependencies of all these types.

−−from−pattern=, **−−to−pattern=**
When used in conjunction with the **dot** command (see above), this selects which relationships are shown in the dependency graph. Both options require a **glob**(7) pattern as an argument, which will be matched against the left−hand and the right−hand, respectively, nodes of a relationship.

Each of these can be used more than once, in which case the unit name must match one of the values. When tests for both sides of the relation are present, a relation must pass both tests to be shown. When patterns are also specified as positional arguments, they must match at least one side of the relation. In other words, patterns specified with those two options will trim the list of edges matched by the positional arguments, if any are given, and fully determine the list of edges shown otherwise.

−−fuzz=*timespan*
When used in conjunction with the **critical−chain** command (see above), also show units, which finished *timespan* earlier, than the latest unit in the same level. The unit of *timespan* is seconds unless specified with a different unit, e.g. "50ms".

−−man=no
Do not invoke **man**(1) to verify the existence of man pages listed in *Documentation*=.

−−generators
Invoke unit generators, see **systemd.generator**(7). Some generators require root privileges. Under a normal user, running with generators enabled will generally result in some warnings.

−−root=*PATH*
With **cat−files**, show config files underneath the specified root path *PATH*.

−−iterations=*NUMBER*
When used with the **calendar** command, show the specified number of iterations the specified calendar expression will elapse next. Defaults to 1.

−−base−time=*TIMESTAMP*
When used with the **calendar** command, show next iterations relative to the specified point in time. If not specified defaults to the current time.

−H, **−−host=**

Execute the operation remotely. Specify a hostname, or a username and hostname separated by "@", to connect to. The hostname may optionally be suffixed by a port ssh is listening on, separated by ":", and then a container name, separated by "/", which connects directly to a specific container on the specified host. This will use SSH to talk to the remote machine manager instance. Container names may be enumerated with **machinectl −H** *HOST*. Put IPv6 addresses in brackets.

−M, **−−machine=**

Execute operation on a local container. Specify a container name to connect to, optionally prefixed by a user name to connect as and a separating "@" character. If the special string ".host" is used in place of the container name, a connection to the local system is made (which is useful to connect to a specific user's user bus: "−−user −−machine=lennart@.host"). If the "@" syntax is not used, the connection is made as root user. If the "@" syntax is used either the left hand side or the right hand side may be omitted (but not both) in which case the local user name and ".host" are implied.

−h, **−−help**

Print a short help text and exit.

−−version

Print a short version string and exit.

−−no−pager

Do not pipe output into a pager.

EXIT STATUS

On success, 0 is returned, a non−zero failure code otherwise.

ENVIRONMENT

$SYSTEMD_LOG_LEVEL

The maximum log level of emitted messages (messages with a higher log level, i.e. less important ones, will be suppressed). Either one of (in order of decreasing importance) **emerg**, **alert**, **crit**, **err**, **warning**, **notice**, **info**, **debug**, or an integer in the range 0...7. See **syslog**(3) for more information.

$SYSTEMD_LOG_COLOR

A boolean. If true, messages written to the tty will be colored according to priority.

This setting is only useful when messages are written directly to the terminal, because **journalctl**(1) and other tools that display logs will color messages based on the log level on their own.

$SYSTEMD_LOG_TIME

A boolean. If true, console log messages will be prefixed with a timestamp.

This setting is only useful when messages are written directly to the terminal or a file, because **journalctl**(1) and other tools that display logs will attach timestamps based on the entry metadata on their own.

$SYSTEMD_LOG_LOCATION

A boolean. If true, messages will be prefixed with a filename and line number in the source code where the message originates.

Note that the log location is often attached as metadata to journal entries anyway. Including it directly in the message text can nevertheless be convenient when debugging programs.

$SYSTEMD_LOG_TID

A boolean. If true, messages will be prefixed with the current numerical thread ID (TID).

Note that the this information is attached as metadata to journal entries anyway. Including it directly in the message text can nevertheless be convenient when debugging programs.

$SYSTEMD_LOG_TARGET

The destination for log messages. One of **console** (log to the attached tty), **console−prefixed** (log to the attached tty but with prefixes encoding the log level and "facility", see **syslog**(3), **kmsg** (log to the kernel circular log buffer), **journal** (log to the journal), **journal−or−kmsg** (log to the journal if available, and to kmsg otherwise), **auto** (determine the appropriate log target automatically, the default), **null** (disable log output).

$SYSTEMD_PAGER
: Pager to use when **−−no−pager** is not given; overrides *$PAGER*. If neither *$SYSTEMD_PAGER* nor *$PAGER* are set, a set of well−known pager implementations are tried in turn, including **less**(1) and **more**(1), until one is found. If no pager implementation is discovered no pager is invoked. Setting this environment variable to an empty string or the value "cat" is equivalent to passing **−−no−pager**.

$SYSTEMD_LESS
: Override the options passed to **less** (by default "FRSXMK").

Users might want to change two options in particular:

K
: This option instructs the pager to exit immediately when Ctrl+C is pressed. To allow **less** to handle Ctrl+C itself to switch back to the pager command prompt, unset this option.

If the value of *$SYSTEMD_LESS* does not include "K", and the pager that is invoked is **less**, Ctrl+C will be ignored by the executable, and needs to be handled by the pager.

X
: This option instructs the pager to not send termcap initialization and deinitialization strings to the terminal. It is set by default to allow command output to remain visible in the terminal even after the pager exits. Nevertheless, this prevents some pager functionality from working, in particular paged output cannot be scrolled with the mouse.

See **less**(1) for more discussion.

$SYSTEMD_LESSCHARSET
: Override the charset passed to **less** (by default "utf−8", if the invoking terminal is determined to be UTF−8 compatible).

$SYSTEMD_PAGERSECURE
: Takes a boolean argument. When true, the "secure" mode of the pager is enabled; if false, disabled. If *$SYSTEMD_PAGERSECURE* is not set at all, secure mode is enabled if the effective UID is not the same as the owner of the login session, see **geteuid**(2) and **sd_pid_get_owner_uid**(3). In secure mode, **LESSSECURE=1** will be set when invoking the pager, and the pager shall disable commands that open or create new files or start new subprocesses. When *$SYSTEMD_PAGERSECURE* is not set at all, pagers which are not known to implement secure mode will not be used. (Currently only **less**(1) implements secure mode.)

Note: when commands are invoked with elevated privileges, for example under **sudo**(8) or **pkexec**(1), care must be taken to ensure that unintended interactive features are not enabled. "Secure" mode for the pager may be enabled automatically as describe above. Setting *SYSTEMD_PAGERSECURE=0* or not removing it from the inherited environment allows the user to invoke arbitrary commands. Note that if the *$SYSTEMD_PAGER* or *$PAGER* variables are to be honoured, *$SYSTEMD_PAGERSECURE* must be set too. It might be reasonable to completely disable the pager using **−−no−pager** instead.

$SYSTEMD_COLORS
: Takes a boolean argument. When true, **systemd** and related utilities will use colors in their output, otherwise the output will be monochrome. Additionally, the variable can take one of the following special values: "16", "256" to restrict the use of colors to the base 16 or 256 ANSI colors, respectively. This can be specified to override the automatic decision based on *$TERM* and what the console is connected to.

$SYSTEMD_URLIFY

The value must be a boolean. Controls whether clickable links should be generated in the output for terminal emulators supporting this. This can be specified to override the decision that **systemd** makes based on *$TERM* and other conditions.

SEE ALSO

systemd(1), **systemctl**(1)