

**NAME**

systemd, init – systemd system and service manager

**SYNOPSIS**

**/lib/systemd/systemd** [OPTIONS...]

**init** [OPTIONS...] {COMMAND}

**DESCRIPTION**

systemd is a system and service manager for Linux operating systems. When run as first process on boot (as PID 1), it acts as init system that brings up and maintains userspace services.

For compatibility with SysV, if systemd is called as **init** and a PID that is not 1, it will execute **telinit** and pass all command line arguments unmodified. That means **init** and **telinit** are mostly equivalent when invoked from normal login sessions. See **telinit(8)** for more information.

When run as a system instance, systemd interprets the configuration file `system.conf` and the files in `system.conf.d` directories; when run as a user instance, systemd interprets the configuration file `user.conf` and the files in `user.conf.d` directories. See **systemd-system.conf(5)** for more information.

**OPTIONS**

The following options are understood:

**--test**

Determine startup sequence, dump it and exit. This is an option useful for debugging only.

**--dump-configuration-items**

Dump understood unit configuration items. This outputs a terse but complete list of configuration items understood in unit definition files.

**--dump-bus-properties**

Dump exposed bus properties. This outputs a terse but complete list of properties exposed to dbus.

**--unit=**

Set default unit to activate on startup. If not specified, defaults to `default.target`.

**--system, --user**

For **--system**, tell systemd to run a system instance, even if the process ID is not 1, i.e. systemd is not run as init process. **--user** does the opposite, running a user instance even if the process ID is 1.

Normally, it should not be necessary to pass these options, as systemd automatically detects the mode it is started in. These options are hence of little use except for debugging. Note that it is not supported booting and maintaining a full system with systemd running in **--system** mode, but PID not 1. In practice, passing **--system** explicitly is only useful in conjunction with **--test**.

**--dump-core**

Enable core dumping on crash. This switch has no effect when running as user instance. This setting may also be enabled during boot on the kernel command line via the `systemd.dump_core=` option, see below.

**--crash-vt=VT**

Switch to a specific virtual console (VT) on crash. Takes a positive integer in the range 1–63, or a boolean argument. If an integer is passed, selects which VT to switch to. If **yes**, the VT kernel messages are written to is selected. If **no**, no VT switch is attempted. This switch has no effect when running as user instance. This setting may also be enabled during boot, on the kernel command line via the `systemd.crash_vt=` option, see below.

**--crash-shell**

Run a shell on crash. This switch has no effect when running as user instance. This setting may also be enabled during boot, on the kernel command line via the `systemd.crash_shell=` option, see below.

**--crash-reboot**

Automatically reboot the system on crash. This switch has no effect when running as user instance. This setting may also be enabled during boot, on the kernel command line via the

*systemd.crash\_reboot*= option, see below.

**--confirm-spawn**

Ask for confirmation when spawning processes. This switch has no effect when run as user instance.

**--show-status=**

Takes a boolean argument or the special value **auto**. If on, terse unit status information is shown on the console during boot-up and shutdown. If off, no such status information is shown. If set to **auto** behavior is similar to off, except that it is automatically switched to on, as soon as the first unit failure or significant boot delay is encountered. This switch has no effect when invoked as user instance. If specified, overrides both the kernel command line setting *systemd.show\_status*= (see below) and the configuration file option **ShowStatus**=, see **systemd-system.conf**(5).

**--log-target=**

Set log target. Argument must be one of **console**, **journal**, **kmsg**, **journal-or-kmsg**, **null**.

**--log-level=**

Set log level. As argument this accepts a numerical log level or the well-known **syslog**(3) symbolic names (lowercase): **emerg**, **alert**, **crit**, **err**, **warning**, **notice**, **info**, **debug**.

**--log-color=**

Highlight important log messages. Argument is a boolean value. If the argument is omitted, it defaults to **true**.

**--log-location=**

Include code location in log messages. This is mostly relevant for debugging purposes. Argument is a boolean value. If the argument is omitted it defaults to **true**.

**--default-standard-output=, --default-standard-error=**

Sets the default output or error output for all services and sockets, respectively. That is, controls the default for **StandardOutput**= and **StandardError**= (see **systemd.exec**(5) for details). Takes one of **inherit**, **null**, **tty**, **journal**, **journal+console**, **syslog**, **syslog+console**, **kmsg**, **kmsg+console**. If the argument is omitted **--default-standard-output**= defaults to **journal** and **--default-standard-error**= to **inherit**.

**--machine-id=**

Override the machine-id set on the hard drive, useful for network booting or for containers. May not be set to all zeros.

**--service-watchdogs=**

Globally enable/disable all service watchdog timeouts and emergency actions. This setting may also be specified during boot, on the kernel command line via the *systemd.service\_watchdogs*= option, see below. Defaults to enabled.

**-h, --help**

Print a short help text and exit.

**--version**

Print a short version string and exit.

## CONCEPTS

systemd provides a dependency system between various entities called "units" of 11 different types. Units encapsulate various objects that are relevant for system boot-up and maintenance. The majority of units are configured in unit configuration files, whose syntax and basic set of options is described in **systemd.unit**(5), however some are created automatically from other configuration, dynamically from system state or programmatically at runtime. Units may be "active" (meaning started, bound, plugged in, ..., depending on the unit type, see below), or "inactive" (meaning stopped, unbound, unplugged, ...), as well as in the process of being activated or deactivated, i.e. between the two states (these states are called "activating", "deactivating"). A special "failed" state is available as well, which is very similar to "inactive" and is entered when the service failed in some way (process returned error code on exit, or crashed, an operation timed out, or after too many restarts). If this state is entered, the cause will be logged, for later reference. Note that the various unit types may have a number of additional substates, which are mapped to

the five generalized unit states described here.

The following unit types are available:

1. Service units, which start and control daemons and the processes they consist of. For details, see **systemd.service(5)**.
2. Socket units, which encapsulate local IPC or network sockets in the system, useful for socket-based activation. For details about socket units, see **systemd.socket(5)**, for details on socket-based activation and other forms of activation, see **daemon(7)**.
3. Target units are useful to group units, or provide well-known synchronization points during boot-up, see **systemd.target(5)**.
4. Device units expose kernel devices in systemd and may be used to implement device-based activation. For details, see **systemd.device(5)**.
5. Mount units control mount points in the file system, for details see **systemd.mount(5)**.
6. Automount units provide automount capabilities, for on-demand mounting of file systems as well as parallelized boot-up. See **systemd.automount(5)**.
7. Timer units are useful for triggering activation of other units based on timers. You may find details in **systemd.timer(5)**.
8. Swap units are very similar to mount units and encapsulate memory swap partitions or files of the operating system. They are described in **systemd.swap(5)**.
9. Path units may be used to activate other services when file system objects change or are modified. See **systemd.path(5)**.
10. Slice units may be used to group units which manage system processes (such as service and scope units) in a hierarchical tree for resource management purposes. See **systemd.slice(5)**.
11. Scope units are similar to service units, but manage foreign processes instead of starting them as well. See **systemd.scope(5)**.

Units are named as their configuration files. Some units have special semantics. A detailed list is available in **systemd.special(7)**.

systemd knows various kinds of dependencies, including positive and negative requirement dependencies (i.e. *Requires=* and *Conflicts=*) as well as ordering dependencies (*After=* and *Before=*). NB: ordering and requirement dependencies are orthogonal. If only a requirement dependency exists between two units (e.g. *foo.service* requires *bar.service*), but no ordering dependency (e.g. *foo.service* after *bar.service*) and both are requested to start, they will be started in parallel. It is a common pattern that both requirement and ordering dependencies are placed between two units. Also note that the majority of dependencies are implicitly created and maintained by systemd. In most cases, it should be unnecessary to declare additional dependencies manually, however it is possible to do this.

Application programs and units (via dependencies) may request state changes of units. In systemd, these requests are encapsulated as 'jobs' and maintained in a job queue. Jobs may succeed or can fail, their execution is ordered based on the ordering dependencies of the units they have been scheduled for.

On boot systemd activates the target unit *default.target* whose job is to activate on-boot services and other on-boot units by pulling them in via dependencies. Usually, the unit name is just an alias (symlink) for either *graphical.target* (for fully-featured boots into the UI) or *multi-user.target* (for limited console-only boots for use in embedded or server environments, or similar; a subset of *graphical.target*). However, it is at the discretion of the administrator to configure it as an alias to any other target unit. See **systemd.special(7)** for details about these target units.

systemd only keeps a minimal set of units loaded into memory. Specifically, the only units that are kept loaded into memory are those for which at least one of the following conditions is true:

1. It is in an active, activating, deactivating or failed state (i.e. in any unit state except for "inactive")

2. It has a job queued for it
3. It is a dependency of some sort of at least one other unit that is loaded into memory
4. It has some form of resource still allocated (e.g. a service unit that is inactive but for which a process is still lingering that ignored the request to be terminated)
5. It has been pinned into memory programmatically by a D-Bus call

systemd will automatically and implicitly load units from disk — if they are not loaded yet — as soon as operations are requested for them. Thus, in many respects, the fact whether a unit is loaded or not is invisible to clients. Use **systemctl list-units --all** to comprehensively list all units currently loaded. Any unit for which none of the conditions above applies is promptly unloaded. Note that when a unit is unloaded from memory its accounting data is flushed out too. However, this data is generally not lost, as a journal log record is generated declaring the consumed resources whenever a unit shuts down.

Processes systemd spawns are placed in individual Linux control groups named after the unit which they belong to in the private systemd hierarchy. (see [cgroups.txt](#)<sup>[1]</sup> for more information about control groups, or short "cgroups"). systemd uses this to effectively keep track of processes. Control group information is maintained in the kernel, and is accessible via the file system hierarchy (beneath `/sys/fs/cgroup/systemd/`), or in tools such as **systemd-cgls(1)** or **ps(1)** (**ps xawf -eo pid,user,cgroup,args** is particularly useful to list all processes and the systemd units they belong to.).

systemd is compatible with the SysV init system to a large degree: SysV init scripts are supported and simply read as an alternative (though limited) configuration file format. The SysV `/dev/initctl` interface is provided, and compatibility implementations of the various SysV client tools are available. In addition to that, various established Unix functionality such as `/etc/fstab` or the `utmp` database are supported.

systemd has a minimal transaction system: if a unit is requested to start up or shut down it will add it and all its dependencies to a temporary transaction. Then, it will verify if the transaction is consistent (i.e. whether the ordering of all units is cycle-free). If it is not, systemd will try to fix it up, and removes non-essential jobs from the transaction that might remove the loop. Also, systemd tries to suppress non-essential jobs in the transaction that would stop a running service. Finally it is checked whether the jobs of the transaction contradict jobs that have already been queued, and optionally the transaction is aborted then. If all worked out and the transaction is consistent and minimized in its impact it is merged with all already outstanding jobs and added to the run queue. Effectively this means that before executing a requested operation, systemd will verify that it makes sense, fixing it if possible, and only failing if it really cannot work.

Note that transactions are generated independently of a unit's state at runtime, hence, for example, if a start job is requested on an already started unit, it will still generate a transaction and wake up any inactive dependencies (and cause propagation of other jobs as per the defined relationships). This is because the enqueued job is at the time of execution compared to the target unit's state and is marked successful and complete when both satisfy. However, this job also pulls in other dependencies due to the defined relationships and thus leads to, in our our example, start jobs for any of those inactive units getting queued as well.

systemd contains native implementations of various tasks that need to be executed as part of the boot process. For example, it sets the hostname or configures the loopback network device. It also sets up and mounts various API file systems, such as `/sys` or `/proc`.

For more information about the concepts and ideas behind systemd, please refer to the [Original Design Document](#)<sup>[2]</sup>.

Note that some but not all interfaces provided by systemd are covered by the [Interface Stability Promise](#)<sup>[3]</sup>.

Units may be generated dynamically at boot and system manager reload time, for example based on other configuration files or parameters passed on the kernel command line. For details, see **systemd.generator(7)**.

Systems which invoke systemd in a container or `initrd` environment should implement the [Container Interface](#)<sup>[4]</sup> or [initrd Interface](#)<sup>[5]</sup> specifications, respectively.

## DIRECTORIES

### System unit directories

The systemd system manager reads unit configuration from various directories. Packages that want to install unit files shall place them in the directory returned by **pkg-config systemd --variable=systemdsystemunitdir**. Other directories checked are `/usr/local/lib/systemd/system` and `/lib/systemd/system`. User configuration always takes precedence. **pkg-config systemd --variable=systemdsystemconfdir** returns the path of the system configuration directory. Packages should alter the content of these directories only with the **enable** and **disable** commands of the **systemctl(1)** tool. Full list of directories is provided in **systemd.unit(5)**.

### User unit directories

Similar rules apply for the user unit directories. However, here the [XDG Base Directory specification](#)<sup>[6]</sup> is followed to find units. Applications should place their unit files in the directory returned by **pkg-config systemd --variable=systemduserunitdir**. Global configuration is done in the directory reported by **pkg-config systemd --variable=systemduserconfdir**. The **enable** and **disable** commands of the **systemctl(1)** tool can handle both global (i.e. for all users) and private (for one user) enabling/disabling of units. Full list of directories is provided in **systemd.unit(5)**.

### SysV init scripts directory

The location of the SysV init script directory varies between distributions. If systemd cannot find a native unit file for a requested service, it will look for a SysV init script of the same name (with the `.service` suffix removed).

### SysV runlevel link farm directory

The location of the SysV runlevel link farm directory varies between distributions. systemd will take the link farm into account when figuring out whether a service shall be enabled. Note that a service unit with a native unit configuration file cannot be started by activating it in the SysV runlevel link farm.

## SIGNALS

### SIGTERM

Upon receiving this signal the systemd system manager serializes its state, reexecutes itself and deserializes the saved state again. This is mostly equivalent to **systemctl daemon-reexec**.

systemd user managers will start the `exit.target` unit when this signal is received. This is mostly equivalent to **systemctl --user start exit.target --job-mode=replace-irreversible**.

### SIGINT

Upon receiving this signal the systemd system manager will start the `ctrl-alt-del.target` unit. This is mostly equivalent to **systemctl start ctrl-alt-del.target --job-mode=replace-irreversible**. If this signal is received more than 7 times per 2s, an immediate reboot is triggered. Note that pressing `Ctrl+Alt+Del` on the console will trigger this signal. Hence, if a reboot is hanging, pressing `Ctrl+Alt+Del` more than 7 times in 2 seconds is a relatively safe way to trigger an immediate reboot.

systemd user managers treat this signal the same way as **SIGTERM**.

### SIGWINCH

When this signal is received the systemd system manager will start the `kbrequest.target` unit. This is mostly equivalent to **systemctl start kbrequest.target**.

This signal is ignored by systemd user managers.

### SIGPWR

When this signal is received the systemd manager will start the `sigpwr.target` unit. This is mostly equivalent to **systemctl start sigpwr.target**.

### SIGUSR1

When this signal is received the systemd manager will try to reconnect to the D-Bus bus.

### SIGUSR2

When this signal is received the systemd manager will log its complete state in human-readable form. The data logged is the same as printed by **systemd-analyze dump**.

#### **SIGHUP**

Reloads the complete daemon configuration. This is mostly equivalent to **systemctl daemon-reload**.

#### **SIGRTMIN+0**

Enters default mode, starts the default.target unit. This is mostly equivalent to **systemctl isolate default.target**.

#### **SIGRTMIN+1**

Enters rescue mode, starts the rescue.target unit. This is mostly equivalent to **systemctl isolate rescue.target**.

#### **SIGRTMIN+2**

Enters emergency mode, starts the emergency.service unit. This is mostly equivalent to **systemctl isolate emergency.service**.

#### **SIGRTMIN+3**

Halts the machine, starts the halt.target unit. This is mostly equivalent to **systemctl start halt.target --job-mode=replace-irreversible**.

#### **SIGRTMIN+4**

Powers off the machine, starts the poweroff.target unit. This is mostly equivalent to **systemctl start poweroff.target --job-mode=replace-irreversible**.

#### **SIGRTMIN+5**

Reboots the machine, starts the reboot.target unit. This is mostly equivalent to **systemctl start reboot.target --job-mode=replace-irreversible**.

#### **SIGRTMIN+6**

Reboots the machine via kexec, starts the kexec.target unit. This is mostly equivalent to **systemctl start kexec.target --job-mode=replace-irreversible**.

#### **SIGRTMIN+13**

Immediately halts the machine.

#### **SIGRTMIN+14**

Immediately powers off the machine.

#### **SIGRTMIN+15**

Immediately reboots the machine.

#### **SIGRTMIN+16**

Immediately reboots the machine with kexec.

#### **SIGRTMIN+20**

Enables display of status messages on the console, as controlled via *systemd.show\_status=1* on the kernel command line.

#### **SIGRTMIN+21**

Disables display of status messages on the console, as controlled via *systemd.show\_status=0* on the kernel command line.

#### **SIGRTMIN+22**

Sets the service manager's log level to "debug", in a fashion equivalent to *systemd.log\_level=debug* on the kernel command line.

#### **SIGRTMIN+23**

Restores the log level to its configured value. The configured value is derived from – in order of priority – the value specified with *systemd.log-level=* on the kernel command line, or the value specified with **LogLevel=** in the configuration file, or the built-in default of "info".

#### **SIGRTMIN+24**

Immediately exits the manager (only available for --user instances).

**SIGRTMIN+26**

Restores the log target to its configured value. The configured value is derived from – in order of priority – the value specified with *systemd.log-target=* on the kernel command line, or the value specified with **LogTarget=** in the configuration file, or the built-in default.

**SIGRTMIN+27, SIGRTMIN+28**

Sets the log target to "console" on **SIGRTMIN+27** (or "kmsg" on **SIGRTMIN+28**), in a fashion equivalent to *systemd.log\_target=console* (or *systemd.log\_target=kmsg* on **SIGRTMIN+28**) on the kernel command line.

**ENVIRONMENT***\$SYSTEMD\_LOG\_LEVEL*

systemd reads the log level from this environment variable. This can be overridden with **--log-level=**.

*\$SYSTEMD\_LOG\_TARGET*

systemd reads the log target from this environment variable. This can be overridden with **--log-target=**.

*\$SYSTEMD\_LOG\_COLOR*

Controls whether systemd highlights important log messages. This can be overridden with **--log-color=**.

*\$SYSTEMD\_LOG\_LOCATION*

Controls whether systemd prints the code location along with log messages. This can be overridden with **--log-location=**.

*\$XDG\_CONFIG\_HOME, \$XDG\_CONFIG\_DIRS, \$XDG\_DATA\_HOME, \$XDG\_DATA\_DIRS*

The systemd user manager uses these variables in accordance to the [XDG Base Directory specification](#)<sup>[6]</sup> to find its configuration.

*\$SYSTEMD\_UNIT\_PATH*

Controls where systemd looks for unit files.

*\$SYSTEMD\_SYSVINIT\_PATH*

Controls where systemd looks for SysV init scripts.

*\$SYSTEMD\_SYSVRCND\_PATH*

Controls where systemd looks for SysV init script runlevel link farms.

*\$SYSTEMD\_COLORS*

The value must be a boolean. Controls whether colorized output should be generated. This can be specified to override the decision that **systemd** makes 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.

*\$LISTEN\_PID, \$LISTEN\_FDS, \$LISTEN\_FDNames*

Set by systemd for supervised processes during socket-based activation. See **sd\_listen\_fds(3)** for more information.

*\$NOTIFY\_SOCKET*

Set by systemd for supervised processes for status and start-up completion notification. See **sd\_notify(3)** for more information.

For further environment variables understood by systemd and its various components, see [Known Environment Variables](#)<sup>[7]</sup>.

**KERNEL COMMAND LINE**

When run as system instance systemd parses a number of kernel command line arguments<sup>[8]</sup>:

*systemd.unit=*, *rd.systemd.unit=*

Overrides the unit to activate on boot. Defaults to `default.target`. This may be used to temporarily boot into a different boot unit, for example `rescue.target` or `emergency.service`. See **systemd.special(7)** for details about these units. The option prefixed with "rd." is honored only in the initial RAM disk (`initrd`), while the one that is not prefixed only in the main system.

#### *systemd.dump\_core*

Takes a boolean argument or enables the option if specified without an argument. If enabled, the systemd manager (PID 1) dumps core when it crashes. Otherwise, no core dump is created. Defaults to enabled.

#### *systemd.crash\_chvt*

Takes a positive integer, or a boolean argument. Can be also specified without an argument, with the same effect as a positive boolean. If a positive integer (in the range 1–63) is specified, the system manager (PID 1) will activate the specified virtual terminal (VT) when it crashes. Defaults to disabled, meaning that no such switch is attempted. If set to enabled, the VT the kernel messages are written to is selected.

#### *systemd.crash\_shell*

Takes a boolean argument or enables the option if specified without an argument. If enabled, the system manager (PID 1) spawns a shell when it crashes, after a 10s delay. Otherwise, no shell is spawned. Defaults to disabled, for security reasons, as the shell is not protected by password authentication.

#### *systemd.crash\_reboot*

Takes a boolean argument or enables the option if specified without an argument. If enabled, the system manager (PID 1) will reboot the machine automatically when it crashes, after a 10s delay. Otherwise, the system will hang indefinitely. Defaults to disabled, in order to avoid a reboot loop. If combined with *systemd.crash\_shell*, the system is rebooted after the shell exits.

#### *systemd.confirm\_spawn*

Takes a boolean argument or a path to the virtual console where the confirmation messages should be emitted. Can be also specified without an argument, with the same effect as a positive boolean. If enabled, the system manager (PID 1) asks for confirmation when spawning processes using `/dev/console`. If a path or a console name (such as "ttyS0") is provided, the virtual console pointed to by this path or described by the given name will be used instead. Defaults to disabled.

#### *systemd.service\_watchdogs=*

Takes a boolean argument. If disabled, all service runtime watchdogs (**WatchdogSec=**) and emergency actions (e.g. **OnFailure=** or **StartLimitAction=**) are ignored by the system manager (PID 1); see **systemd.service(5)**. Defaults to enabled, i.e. watchdogs and failure actions are processed normally. The hardware watchdog is not affected by this option.

#### *systemd.show\_status*

Takes a boolean argument or the constant **auto**. Can be also specified without an argument, with the same effect as a positive boolean. If enabled, the systemd manager (PID 1) shows terse service status updates on the console during bootup. **auto** behaves like **false** until a unit fails or there is a significant delay in boot. Defaults to enabled, unless **quiet** is passed as kernel command line option, in which case it defaults to **auto**. If specified overrides the system manager configuration file option **ShowStatus=**, see **systemd-system.conf(5)**. However, the process command line option **--show-status=** takes precedence over both this kernel command line option and the configuration file option.

#### *systemd.log\_target=, systemd.log\_level=, systemd.log\_location=, systemd.log\_color*

Controls log output, with the same effect as the `$SYSTEMD_LOG_TARGET`, `$SYSTEMD_LOG_LEVEL`, `$SYSTEMD_LOG_LOCATION`, `$SYSTEMD_LOG_COLOR` environment variables described above. *systemd.log\_color* can be specified without an argument, with the same effect as a positive boolean.

#### *systemd.default\_standard\_output=, systemd.default\_standard\_error=*

Controls default standard output and error output for services, with the same effect as the



**--default-standard-output=** and **--default-standard-error=** command line arguments described above, respectively.

*systemd.setenv=*

Takes a string argument in the form `VARIABLE=VALUE`. May be used to set default environment variables to add to forked child processes. May be used more than once to set multiple variables.

*systemd.machine\_id=*

Takes a 32 character hex value to be used for setting the machine-id. Intended mostly for network booting where the same machine-id is desired for every boot.

*systemd.unified\_cgroup\_hierarchy*

When specified without an argument or with a true argument, enables the usage of **unified cgroup hierarchy**<sup>[9]</sup> (a.k.a. `cgroups-v2`). When specified with a false argument, fall back to hybrid or full legacy cgroup hierarchy.

If this option is not specified, the default behaviour is determined during compilation (the **--default-hierarchy=** meson option). If the kernel does not support unified cgroup hierarchy, the legacy hierarchy will be used even if this option is specified.

*systemd.legacy\_systemd\_cgroup\_controller*

Takes effect if the full unified cgroup hierarchy is not used (see previous option). When specified without an argument or with a true argument, disables the use of "hybrid" cgroup hierarchy (i.e. a `cgroups-v2` tree used for systemd, and **legacy cgroup hierarchy**<sup>[10]</sup>, a.k.a. `cgroups-v1`, for other controllers), and forces a full "legacy" mode. When specified with a false argument, enables the use of "hybrid" hierarchy.

If this option is not specified, the default behaviour is determined during compilation (the **--default-hierarchy=** meson option). If the kernel does not support unified cgroup hierarchy, the legacy hierarchy will be used even if this option is specified.

*quiet*

Turn off status output at boot, much like *systemd.show\_status=no* would. Note that this option is also read by the kernel itself and disables kernel log output. Passing this option hence turns off the usual output from both the system manager and the kernel.

*debug*

Turn on debugging output. This is equivalent to *systemd.log\_level=debug*. Note that this option is also read by the kernel itself and enables kernel debug output. Passing this option hence turns on the debug output from both the system manager and the kernel.

*emergency, rd.emergency, -b*

Boot into emergency mode. This is equivalent to *systemd.unit=emergency.target* or *rd.systemd.unit=emergency.target*, respectively, and provided for compatibility reasons and to be easier to type.

*rescue, rd.rescue, single, s, S, 1*

Boot into rescue mode. This is equivalent to *systemd.unit=rescue.target* or *rd.systemd.unit=rescue.target*, respectively, and provided for compatibility reasons and to be easier to type.

*2, 3, 4, 5*

Boot into the specified legacy SysV runlevel. These are equivalent to *systemd.unit=runlevel2.target*, *systemd.unit=runlevel3.target*, *systemd.unit=runlevel4.target*, and *systemd.unit=runlevel5.target*, respectively, and provided for compatibility reasons and to be easier to type.

*locale.LANG=, locale.LANGUAGE=, locale.LC\_CTYPE=, locale.LC\_NUMERIC=, locale.LC\_TIME=, locale.LC\_COLLATE=, locale.LC\_MONETARY=, locale.LC\_MESSAGES=, locale.LC\_PAPER=, locale.LC\_NAME=, locale.LC\_ADDRESS=, locale.LC\_TELEPHONE=, locale.LC\_MEASUREMENT=, locale.LC\_IDENTIFICATION=*

Set the system locale to use. This overrides the settings in `/etc/locale.conf`. For more information, see **locale.conf(5)** and **locale(7)**.

For other kernel command line parameters understood by components of the core OS, please refer to **kernel-command-line(7)**.

## SOCKETS AND FIFOS

`/run/systemd/notify`

Daemon status notification socket. This is an **AF\_UNIX** datagram socket and is used to implement the daemon notification logic as implemented by **sd\_notify(3)**.

`/run/systemd/private`

Used internally as communication channel between **systemctl(1)** and the `systemd` process. This is an **AF\_UNIX** stream socket. This interface is private to `systemd` and should not be used in external projects.

`/dev/initctl`

Limited compatibility support for the SysV client interface, as implemented by the `systemd-initctl.service` unit. This is a named pipe in the file system. This interface is obsolete and should not be used in new applications.

## SEE ALSO

The [systemd Homepage](#)<sup>[11]</sup>, **systemd-system.conf(5)**, **locale.conf(5)**, **systemctl(1)**, **journalctl(1)**, **systemd-notify(1)**, **daemon(7)**, **sd-daemon(3)**, **systemd.unit(5)**, **systemd.special(5)**, **pkg-config(1)**, **kernel-command-line(7)**, **bootup(7)**, **systemd.directives(7)**

## NOTES

1. `cgroups.txt`  
<https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt>
2. Original Design Document  
<http://0pointer.de/blog/projects/systemd.html>
3. Interface Stability Promise  
<https://www.freedesktop.org/wiki/Software/systemd/InterfaceStabilityPromise>
4. Container Interface  
<https://www.freedesktop.org/wiki/Software/systemd/ContainerInterface>
5. `initrd` Interface  
<https://www.freedesktop.org/wiki/Software/systemd/InitrdInterface>
6. XDG Base Directory specification  
<http://standards.freedesktop.org/basedir-spec/basedir-spec-latest.html>
7. Known Environment Variables  
<https://systemd.io/ENVIRONMENT>
8. If run inside a Linux container these arguments may be passed as command line arguments to `systemd` itself, next to any of the command line options listed in the Options section above. If run outside of Linux containers, these arguments are parsed from `/proc/cmdline` instead.
9. unified cgroup hierarchy  
<https://www.kernel.org/doc/Documentation/cgroup-v2.txt>
10. legacy cgroup hierarchy  
<https://www.kernel.org/doc/Documentation/cgroup-v1/>
11. `systemd` Homepage  
<https://www.freedesktop.org/wiki/Software/systemd/>