Name

systemctl — Control the systemd system and service manager

Synopsis

`systemctl` [OPTIONS...] COMMAND [UNIT...]

Description

systemctl may be used to introspect and control the state of the "`systemd`" system and service manager. Please refer to systemd(1) for an introduction into the basic concepts and functionality this tool manages.

Options

The following options are understood:

`-t, --type=`

The argument should be a comma-separated list of unit types such as `service` and `socket`.

If one of the arguments is a unit type, when listing units, limit display to certain unit types. Otherwise, units of all types will be shown.

As a special case, if one of the arguments is `help`, a list of allowed values will be printed and the program will exit.

`--state=`

The argument should be a comma-separated list of unit LOAD, SUB, or ACTIVE states. When listing units, show only those in the specified states. Use `--state=failed` to show only failed units.

As a special case, if one of the arguments is `help`, a list of allowed values will be printed and the program will exit.

`-p, --property=`

When showing unit/job/manager properties with the **show** command, limit display to properties specified in the argument. The argument should be a comma-separated list of property names, such as "`MainPID`". Unless specified, all known properties are shown. If specified more than once, all properties with the specified names are shown. Shell completion is implemented for property names.

For the manager itself, **systemctl show** will show all available properties. Those properties are documented in systemd-system.conf(5).

Properties for units vary by unit type, so showing any unit (even a non-existent one) is a way to list properties pertaining to this type. Similarly, showing any job will list properties pertaining to all jobs. Properties for units are documented in systemd.unit(5), and the pages for individual unit types systemd.service(5), systemd.socket(5), etc.

`-a, --all`

When listing units with **list-units**, also show inactive units and units which are following other units. When showing unit/job/manager properties, show all properties regardless whether they are set or not.

To list all units installed in the file system, use the **list-unit-files** command instead.

When listing units with **list-dependencies**, recursively show dependencies of all dependent units (by default only dependencies of target units are shown).

When used with **status**, show journal messages in full, even if they include unprintable characters or are very long. By default, fields with unprintable characters are abbreviated as "blob data". (Note that the pager may escape unprintable characters again.)

`-r, --recursive`

When listing units, also show units of local containers. Units of local containers will be prefixed with the container name, separated by a single colon character ("`:`").

`--reverse`

Show reverse dependencies between units with **list-dependencies**, i.e. follow dependencies of type `WantedBy=`, `RequiredBy=`, `PartOf=`, `BoundBy=`, instead of `Wants=` and similar.

`--after`

With **list-dependencies**, show the units that are ordered before the specified unit. In other words, recursively list units following the `After=` dependency.

Note that any `After=` dependency is automatically mirrored to create a `Before=` dependency. Temporal dependencies may be specified explicitly, but are also created implicitly for units which are `WantedBy=` targets (see systemd.target(5)), and as a result

of other directives (for example `RequiresMountsFor=`). Both explicitly and implicitly introduced dependencies are shown with **list-dependencies**.

When passed to the **list-jobs** command, for each printed job show which other jobs are waiting for it. May be combined with `--before` to show both the jobs waiting for each job as well as all jobs each job is waiting for.

`--before`

With **list-dependencies**, show the units that are ordered after the specified unit. In other words, recursively list units following the `Before=` dependency.

When passed to the **list-jobs** command, for each printed job show which other jobs it is waiting for. May be combined with `--after` to show both the jobs waiting for each job as well as all jobs each job is waiting for.

`-l, --full`

Do not ellipsize unit names, process tree entries, journal output, or truncate unit descriptions in the output of **status**, **list-units**, **list-jobs**, and **list-timers**.

Also, show installation targets in the output of **is-enabled**.

`--value`

When printing properties with **show**, only print the value, and skip the property name and "`=`".

`--show-types`

When showing sockets, show the type of the socket.

`--job-mode=`

When queuing a new job, this option controls how to deal with already queued jobs. It takes one of "`fail`", "`replace`", "`replace-irreversibly`", "`isolate`", "`ignore-dependencies`", "`ignore-requirements`" or "`flush`". Defaults to "`replace`", except when the **isolate** command is used which implies the "`isolate`" job mode.

If "`fail`" is specified and a requested operation conflicts with a pending job (more specifically: causes an already pending start job to be reversed into a stop job or vice versa), cause the operation to fail.

If "`replace`" (the default) is specified, any conflicting pending job will be replaced, as necessary.

If "`replace-irreversibly`" is specified, operate like "`replace`", but also mark the new jobs as irreversible. This prevents future conflicting transactions from replacing these jobs (or even being enqueued while the irreversible jobs are still pending). Irreversible jobs can still be cancelled using the **cancel** command. This job mode should be used on any transaction which pulls in `shutdown.target`.

"`isolate`" is only valid for start operations and causes all other units to be stopped when the specified unit is started. This mode is always used when the **isolate** command is used.

"`flush`" will cause all queued jobs to be canceled when the new job is enqueued.

If "`ignore-dependencies`" is specified, then all unit dependencies are ignored for this new job and the operation is executed immediately. If passed, no required units of the unit passed will be pulled in, and no ordering dependencies will be honored. This is mostly a debugging and rescue tool for the administrator and should not be used by applications.

"`ignore-requirements`" is similar to "`ignore-dependencies`", but only causes the requirement dependencies to be ignored, the ordering dependencies will still be honored.

`-T, --show-transaction`

When enqueuing a unit job (for example as effect of a **systemctl start** invocation or similar), show brief information about all jobs enqueued, covering both the requested job and any added because of unit dependencies. Note that the output will only include jobs immediately part of the transaction requested. It is possible that service start-up program code run as effect of the enqueued jobs might request further jobs to be pulled in. This means that completion of the listed jobs might ultimately entail more jobs than the listed ones.

`--fail`

Shorthand for `--job-mode=`fail.

When used with the **kill** command, if no units were killed, the operation results in an error.

`-i, --ignore-inhibitors`

When system shutdown or a sleep state is requested, ignore inhibitor locks. Applications can establish inhibitor locks to avoid that certain important operations (such as CD burning or suchlike) are interrupted by system shutdown or a sleep state. Any user may take these locks and privileged users may override these locks. If any locks are taken, shutdown and sleep state requests will normally fail (unless privileged) and a list of active locks is printed. However, if `--ignore-inhibitors` is specified, the established locks are ignored and not shown, and the operation attempted anyway, possibly requiring additional privileges.

`--dry-run`

Just print what would be done. Currently supported by verbs **halt**, **poweroff**, **reboot**, **kexec**, **suspend**, **hibernate**, **hybrid-sleep**, **suspend-then-hibernate**, **default**, **rescue**, **emergency**, and **exit**.

`-q`, `--quiet`

Suppress printing of the results of various commands and also the hints about truncated log lines. This does not suppress output of commands for which the printed output is the only result (like **show**). Errors are always printed.

`--no-block`

Do not synchronously wait for the requested operation to finish. If this is not specified, the job will be verified, enqueued and **systemctl** will wait until the unit's start-up is completed. By passing this argument, it is only verified and enqueued. This option may not be combined with `--wait`.

`--wait`

Synchronously wait for started units to terminate again. This option may not be combined with `--no-block`. Note that this will wait forever if any given unit never terminates (by itself or by getting stopped explicitly); particularly services which use "`RemainAfterExit=yes`".

When used with **is-system-running**, wait until the boot process is completed before returning.

`--user`

Talk to the service manager of the calling user, rather than the service manager of the system.

`--system`

Talk to the service manager of the system. This is the implied default.

`--failed`

List units in failed state. This is equivalent to `--state=failed`.

`--no-wall`

Do not send wall message before halt, power-off and reboot.

`--global`

When used with **enable** and **disable**, operate on the global user configuration directory, thus enabling or disabling a unit file globally for all future logins of all users.

`--no-reload`

When used with **enable** and **disable**, do not implicitly reload daemon configuration after executing the changes.

`--no-ask-password`

When used with **start** and related commands, disables asking for passwords. Background services may require input of a password or passphrase string, for example to unlock system hard disks or cryptographic certificates. Unless this option is specified and the command is invoked from a terminal, **systemctl** will query the user on the terminal for the necessary secrets. Use this option to switch this behavior off. In this case, the password must be supplied by some other means (for example graphical password agents) or the service might fail. This also disables querying the user for authentication for privileged operations.

`--kill-who=`

When used with **kill**, choose which processes to send a signal to. Must be one of `main`, `control` or `all` to select whether to kill only the main process, the control process or all processes of the unit. The main process of the unit is the one that defines the life-time of it. A control process of a unit is one that is invoked by the manager to induce state changes of it. For example, all processes started due to the `ExecStartPre=`, `ExecStop=` or `ExecReload=` settings of service units are control processes. Note that there is only one control process per unit at a time, as only one state change is executed at a time. For services of type `Type=forking`, the initial process started by the manager for `ExecStart=` is a control process, while the process ultimately forked off by that one is then considered the main process of the unit (if it can be determined). This is different for service units of other types, where the process forked off by the manager for `ExecStart=` is always the main process itself. A service unit consists of zero or one main process, zero or one control process plus any number of additional processes. Not all unit types manage processes of these types however. For example, for mount units, control processes are defined (which are the invocations of `/usr/bin/mount` and `/usr/bin/umount`), but no main process is defined. If omitted, defaults to `all`.

`-s`, `--signal=`

When used with **kill**, choose which signal to send to selected processes. Must be one of the well-known signal specifiers such as `SIGTERM`, `SIGINT` or `SIGSTOP`. If omitted, defaults to `SIGTERM`.

`--what=`

Select what type of per-unit resources to remove when the **clean** command is invoked, see below. Takes one of `configuration`, `state`, `cache`, `logs`, `runtime` to select the type of resource. This option may be specified more than once, in which case all specified resource types are removed. Also accepts the special value `all` as a shortcut for specifiying all five resource types. If this option is not specified defaults to the combination of `cache` and `runtime`, i.e. the two kinds of resources that are generally considered to be redundant and can be reconstructed on next invocation.

`-f`, `--force`

When used with **enable**, overwrite any existing conflicting symlinks.

When used with **edit**, create all of the specified units which do not already exist.

When used with **halt**, **poweroff**, **reboot** or **kexec**, execute the selected operation without shutting down all units. However, all processes will be killed forcibly and all file systems are unmounted or remounted read-only. This is hence a drastic but relatively safe option to request an immediate reboot. If `--force` is specified twice for these operations (with the exception of **kexec**), they will be executed immediately, without terminating any processes or unmounting any file systems. Warning: specifying `--force` twice with any of these operations might result in data loss. Note that when `--force` is specified twice the selected operation is executed by **systemctl** itself, and the system manager is not contacted. This means the command should succeed even when the system manager has crashed.

`--message=`

When used with **halt**, **poweroff** or **reboot**, set a short message explaining the reason for the operation. The message will be logged together with the default shutdown message.

`--now`

When used with **enable**, the units will also be started. When used with **disable** or **mask**, the units will also be stopped. The start or stop operation is only carried out when the respective enable or disable operation has been successful.

`--root=`

When used with **enable**/**disable**/**is-enabled** (and related commands), use the specified root path when looking for unit files. If this option is present, **systemctl** will operate on the file system directly, instead of communicating with the **systemd** daemon to carry out changes.

`--runtime`

When used with **enable**, **disable**, **edit**, (and related commands), make changes only temporarily, so that they are lost on the next reboot. This will have the effect that changes are not made in subdirectories of `/etc` but in `/run`, with identical immediate effects, however, since the latter is lost on reboot, the changes are lost too.

Similarly, when used with **set-property**, make changes only temporarily, so that they are lost on the next reboot.

`--preset-mode=`

Takes one of "`full`" (the default), "`enable-only`", "`disable-only`". When used with the **preset** or **preset-all** commands, controls whether units shall be disabled and enabled according to the preset rules, or only enabled, or only disabled.

`-n, --lines=`

When used with **status**, controls the number of journal lines to show, counting from the most recent ones. Takes a positive integer argument, or 0 to disable journal output. Defaults to 10.

`-o, --output=`

When used with **status**, controls the formatting of the journal entries that are shown. For the available choices, see journalctl(1). Defaults to "`short`".

`--firmware-setup`

When used with the **reboot** command, indicate to the system's firmware to reboot into the firmware setup interface. Note that this functionality is not available on all systems.

`--boot-loader-menu=`

When used with the **reboot** command, indicate to the system's boot loader to show the boot loader menu on the following boot. Takes a time value as parameter — indicating the menu time-out. Pass zero in order to disable the menu time-out. Note that not all boot loaders support this functionality.

`--boot-loader-entry=`

When used with the **reboot** command, indicate to the system's boot loader to boot into a specific boot loader entry on the following boot. Takes a boot loader entry identifier as argument, or "`help`" in order to list available entries. Note that not all boot loaders support this functionality.

`--plain`

When used with **list-dependencies**, **list-units** or **list-machines**, the output is printed as a list instead of a tree, and the bullet circles are omitted.

`-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.

`--no-pager`

Do not pipe output into a pager.

`--no-legend`

Do not print the legend, i.e. column headers and the footer with hints.

`-h`, `--help`

Print a short help text and exit.

`--version`

Print a short version string and exit.

Commands

The following commands are understood:

Unit Commands

list-units [*PATTERN*…]

List units that **systemd** currently has in memory. This includes units that are either referenced directly or through a dependency, units that are pinned by applications programmatically, or units that were active in the past and have failed. By default only units which are active, have pending jobs, or have failed are shown; this can be changed with option `--all`. If one or more *PATTERN*s are specified, only units matching one of them are shown. The units that are shown are additionally filtered by `--type=` and `--state=` if those options are specified.

Produces output similar to

```
  UNIT                         LOAD   ACTIVE SUB     DESCRIPTION
  sys-module-fuse.device       loaded active plugged /sys/module/fuse
  -.mount                      loaded active mounted Root Mount
  boot-efi.mount               loaded active mounted /boot/efi
  systemd-journald.service     loaded active running Journal Service
  systemd-logind.service       loaded active running Login Service
● user@1000.service            loaded failed failed  User Manager for UID 1000
  …
  systemd-tmpfiles-clean.timer loaded active waiting Daily Cleanup of Temporary Directories

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

123 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

The header and the last unit of a given type are underlined if the terminal supports that. A colored dot is shown next to services which were masked, not found, or otherwise failed.

The LOAD column shows the load state, one of `loaded`, `not-found`, `bad-setting`, `error`, `masked`. The ACTIVE columns shows the general unit state, one of `active`, `reloading`, `inactive`, `failed`, `activating`, `deactivating`. The SUB column shows the unit-type-specific detailed state of the unit, possible values vary by unit type. The list of possible LOAD, ACTIVE, and SUB states is not constant and new systemd releases may both add and remove values.

```
systemctl --state=help
```

command maybe be used to display the current set of possible values.

This is the default command.

list-sockets [*PATTERN*…]

List socket units currently in memory, ordered by listening address. If one or more *PATTERN*s are specified, only socket units matching one of them are shown. Produces output similar to

```
LISTEN           UNIT                        ACTIVATES
/dev/initctl     systemd-initctl.socket      systemd-initctl.service
…
[::]:22          sshd.socket                 sshd.service
kobject-uevent 1 systemd-udevd-kernel.socket systemd-udevd.service

5 sockets listed.
```

Note: because the addresses might contains spaces, this output is not suitable for programmatic consumption.

Also see `--show-types`, `--all`, and `--state=`.

list-timers [*PATTERN*…]

List timer units currently in memory, ordered by the time they elapse next. If one or more *PATTERN*s are specified, only units matching one of them are shown. Produces output similar to

```
NEXT                         LEFT          LAST                         PASSED     UNIT                         ACTIVATES
n/a                          n/a           Thu 2017-02-23 13:40:29 EST  3 days ago ureadahead-stop.timer        ureadahead-stop.service
Sun 2017-02-26 18:55:42 EST  1min 14s left Thu 2017-02-23 13:54:44 EST  3 days ago systemd-tmpfiles-clean.timer systemd-tmpfiles-clean.service
Sun 2017-02-26 20:37:16 EST  1h 42min left Sun 2017-02-26 11:56:36 EST  6h ago     apt-daily.timer              apt-daily.service
```

```
Sun 2017-02-26 20:57:49 EST  2h 3min left  Sun 2017-02-26 11:56:36 EST  6h ago     snapd.refresh.timer          snapd.refresh.service
```

NEXT shows the next time the timer will run.

LEFT shows how long till the next time the timer runs.

LAST shows the last time the timer ran.

PASSED shows how long has passed since the timer last ran.

UNIT shows the name of the timer

ACTIVATES shows the name the service the timer activates when it runs.

Also see `--all` and `--state=`.

start ***PATTERN*...**

Start (activate) one or more units specified on the command line.

Note that glob patterns operate on the set of primary names of units currently in memory. Units which are not active and are not in a failed state usually are not in memory, and will not be matched by any pattern. In addition, in case of instantiated units, systemd is often unaware of the instance name until the instance has been started. Therefore, using glob patterns with **start** has limited usefulness. Also, secondary alias names of units are not considered.

stop ***PATTERN*...**

Stop (deactivate) one or more units specified on the command line.

This command will fail if the unit does exist or if stopping of the unit is prohibited (see `RefuseManualStop=` in systemd.unit(5)). It will *not* fail if any of the commands configured to stop the unit (`ExecStop=`, etc.) fail, because the manager will still forcibly terminate the unit.

reload ***PATTERN*...**

Asks all units listed on the command line to reload their configuration. Note that this will reload the service-specific configuration, not the unit configuration file of systemd. If you want systemd to reload the configuration file of a unit, use the **daemon-reload** command. In other words: for the example case of Apache, this will reload Apache's `httpd.conf` in the web server, not the `apache.service` systemd unit file.

This command should not be confused with the **daemon-reload** command.

restart ***PATTERN*...**

Stop and then start one or more units specified on the command line. If the units are not running yet, they will be started.

Note that restarting a unit with this command does not necessarily flush out all of the unit's resources before it is started again. For example, the per-service file descriptor storage facility (see `FileDescriptorStoreMax=` in systemd.service(5)) will remain intact as long as the unit has a job pending, and is only cleared when the unit is fully stopped and no jobs are pending anymore. If it is intended that the file descriptor store is flushed out, too, during a restart operation an explicit **systemctl stop** command followed by **systemctl start** should be issued.

try-restart ***PATTERN*...**

Stop and then start one or more units specified on the command line if the units are running. This does nothing if units are not running.

reload-or-restart ***PATTERN*...**

Reload one or more units if they support it. If not, stop and then start them instead. If the units are not running yet, they will be started.

try-reload-or-restart ***PATTERN*...**

Reload one or more units if they support it. If not, stop and then start them instead. This does nothing if the units are not running.

isolate ***UNIT***

Start the unit specified on the command line and its dependencies and stop all others, unless they have `IgnoreOnIsolate=yes` (see systemd.unit(5)). If a unit name with no extension is given, an extension of "`.target`" will be assumed.

This is similar to changing the runlevel in a traditional init system. The **isolate** command will immediately stop processes that are not enabled in the new unit, possibly including the graphical environment or terminal you are currently using.

Note that this is allowed only on units where `AllowIsolate=` is enabled. See systemd.unit(5) for details.

kill ***PATTERN*...**

Send a signal to one or more processes of the unit. Use `--kill-who=` to select which process to kill. Use `--signal=` to select the signal to send.

clean ***PATTERN*...**

Remove the configuration, state, cache, logs or runtime data of the specified units. Use `--what=` to select which kind of resource to remove. For service units this may be used to remove the directories configured with `ConfigurationDirectory=`, `StateDirectory=`, `CacheDirectory=`, `LogsDirectory=` and `RuntimeDirectory=`, see systemd.exec(5) for details. For timer units this may be used to clear out the persistent timestamp data if `Persistent=` is used and `--what=state` is selected, see systemd.timer(5). This command only applies to units that use either of these settings. If `--what=` is not specified, both the cache and runtime data are removed (as these two types of data are generally redundant and reproducible on the next invocation of the unit).

is-active ***PATTERN*...**

Check whether any of the specified units are active (i.e. running). Returns an exit code `0` if at least one is active, or non-zero otherwise. Unless `--quiet` is specified, this will also print the current unit state to standard output.

is-failed ***PATTERN*...**

Check whether any of the specified units are in a "failed" state. Returns an exit code `0` if at least one has failed, non-zero otherwise. Unless `--quiet` is specified, this will also print the current unit state to standard output.

status [*PATTERN*...|*PID*...]]

Show terse runtime status information about one or more units, followed by most recent log data from the journal. If no units are specified, show system status. If combined with `--all`, also show the status of all units (subject to limitations specified with `-t`). If a PID is passed, show information about the unit the process belongs to.

This function is intended to generate human-readable output. If you are looking for computer-parsable output, use **show** instead. By default, this function only shows 10 lines of output and ellipsizes lines to fit in the terminal window. This can be changed with `--lines` and `--full`, see above. In addition, **journalctl --unit=*NAME*** or **journalctl --user-unit=*NAME*** use a similar filter for messages and might be more convenient.

systemd implicitly loads units as necessary, so just running the **status** will attempt to load a file. The command is thus not useful for determining if something was already loaded or not. The units may possibly also be quickly unloaded after the operation is completed if there's no reason to keep it in memory thereafter.

Example 1. Example output from systemctl status

```
$ systemctl status bluetooth
● bluetooth.service - Bluetooth service
   Loaded: loaded (/usr/lib/systemd/system/bluetooth.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2017-01-04 13:54:04 EST; 1 weeks 0 days ago
     Docs: man:bluetoothd(8)
 Main PID: 930 (bluetoothd)
   Status: "Running"
    Tasks: 1
   Memory: 648.0K
      CPU: 435ms
   CGroup: /system.slice/bluetooth.service
           └─930 /usr/lib/bluetooth/bluetoothd

Jan 12 10:46:45 example.com bluetoothd[8900]: Not enough free handles to register service
Jan 12 10:46:45 example.com bluetoothd[8900]: Current Time Service could not be registered
Jan 12 10:46:45 example.com bluetoothd[8900]: gatt-time-server: Input/output error (5)
```

The dot ("●") uses color on supported terminals to summarize the unit state at a glance. White indicates an "`inactive`" or "`deactivating`" state. Red indicates a "`failed`" or "`error`" state and green indicates an "`active`", "`reloading`" or "`activating`" state.

The "Loaded:" line in the output will show "`loaded`" if the unit has been loaded into memory. Other possible values for "Loaded:" include: "`error`" if there was a problem loading it, "`not-found`" if not unit file was found for this unit, "`bad-setting`" if an essential unit file setting could not be parsed and "`masked`" if the unit file has been masked. Along with showing the path to the unit file, this line will also show the enablement state. Enabled commands start at boot. See the full table of possible enablement states — including the definition of "`masked`" — in the documentation for the **is-enabled** command.

The "Active:" line shows active state. The value is usually "`active`" or "`inactive`". Active could mean started, bound, plugged in, etc depending on the unit type. The unit could also be in process of changing states, reporting a state of "`activating`" or "`deactivating`". A special "`failed`" state is entered when the service failed in some way, such as a crash, exiting with an error code or timing out. If the failed state is entered the cause will be logged for later reference.

show [*PATTERN*...|*JOB*...]

Show properties of one or more units, jobs, or the manager itself. If no argument is specified, properties of the manager will be shown. If a unit name is specified, properties of the unit are shown, and if a job ID is specified, properties of the job are shown. By default, empty properties are suppressed. Use `--all` to show those too. To select specific properties to show, use `--property=`. This command is intended to be used whenever computer-parsable output is required. Use **status** if you are looking for formatted human-readable output.

Many properties shown by **systemctl show** map directly to configuration settings of the system and service manager and its unit files. Note that the properties shown by the command are generally more low-level, normalized versions of the original configuration settings and expose runtime state in addition to configuration. For example, properties shown for service units include the service's current main process identifier as "`MainPID`" (which is runtime state), and time settings are always exposed as properties ending in the "`…USec`" suffix even if a matching configuration options end in "`…Sec`", because microseconds is the normalized time unit used by the system and service manager.

cat ***PATTERN*...**

Show backing files of one or more units. Prints the "fragment" and "drop-ins" (source files) of units. Each file is preceded by a comment which includes the file name. Note that this shows the contents of the backing files on disk, which may not match

the system manager's understanding of these units if any unit files were updated on disk and the **daemon-reload** command wasn't issued since.

set-property ***UNIT PROPERTY=VALUE...***

Set the specified unit properties at runtime where this is supported. This allows changing configuration parameter properties such as resource control settings at runtime. Not all properties may be changed at runtime, but many resource control settings (primarily those in systemd.resource-control(5)) may. The changes are applied immediately, and stored on disk for future boots, unless `--runtime` is passed, in which case the settings only apply until the next reboot. The syntax of the property assignment follows closely the syntax of assignments in unit files.

Example: **systemctl set-property foobar.service CPUWeight=200**

If the specified unit appears to be inactive, the changes will be only stored on disk as described previously hence they will be effective when the unit will be started.

Note that this command allows changing multiple properties at the same time, which is preferable over setting them individually.

Example: **systemctl set-property foobar.service CPUWeight=200 MemoryMax=2G IPAccounting=yes**

Like with unit file configuration settings, assigning an empty setting usually resets a property to its defaults.

Example: **systemctl set-property avahi-daemon.service IPAddressDeny=**

help ***PATTERN...|PID...***

Show manual pages for one or more units, if available. If a PID is given, the manual pages for the unit the process belongs to are shown.

reset-failed [*PATTERN...***]**

Reset the "`failed`" state of the specified units, or if no unit name is passed, reset the state of all units. When a unit fails in some way (i.e. process exiting with non-zero error code, terminating abnormally or timing out), it will automatically enter the "`failed`" state and its exit code and status is recorded for introspection by the administrator until the service is stopped/re-started or reset with this command.

In addition to resetting the "`failed`" state of a unit it also resets various other per-unit properties: the start rate limit counter of all unit types is reset to zero, as is the restart counter of service units. Thus, if a unit's start limit (as configured with `StartLimitIntervalSec=`/`StartLimitBurst=`) is hit and the unit refuses to be started again, use this command to make it startable again.

list-dependencies [*UNIT*]

Shows units required and wanted by the specified unit. This recursively lists units following the `Requires=`, `Requisite=`, `ConsistsOf=`, `Wants=`, `BindsTo=` dependencies. If no unit is specified, `default.target` is implied.

By default, only target units are recursively expanded. When `--all` is passed, all other units are recursively expanded as well.

Options `--reverse`, `--after`, `--before` may be used to change what types of dependencies are shown.

Note that this command only lists units currently loaded into memory by the service manager. In particular, this command is not suitable to get a comprehensive list at all reverse dependencies on a specific unit, as it won't list the dependencies declared by units currently not loaded.

Unit File Commands

list-unit-files [*PATTERN…*]

List unit files installed on the system, in combination with their enablement state (as reported by **is-enabled**). If one or more *PATTERN*s are specified, only unit files whose name matches one of them are shown (patterns matching unit file system paths are not supported).

enable ***UNIT...***, **enable** ***PATH...***

Enable one or more units or unit instances. This will create a set of symlinks, as encoded in the "`[Install]`" sections of the indicated unit files. After the symlinks have been created, the system manager configuration is reloaded (in a way equivalent to **daemon-reload**), in order to ensure the changes are taken into account immediately. Note that this does *not* have the effect of also starting any of the units being enabled. If this is desired, combine this command with the `--now` switch, or invoke **start** with appropriate arguments later. Note that in case of unit instance enablement (i.e. enablement of units of the form `foo@bar.service`), symlinks named the same as instances are created in the unit configuration directory, however they point to the single template unit file they are instantiated from.

This command expects either valid unit names (in which case various unit file directories are automatically searched for unit files with appropriate names), or absolute paths to unit files (in which case these files are read directly). If a specified unit file is located outside of the usual unit file directories, an additional symlink is created, linking it into the unit configuration path, thus ensuring it is found when requested by commands such as **start**. The file system where the linked unit files are located must be accessible when systemd is started (e.g. anything underneath `/home` or `/var` is not allowed, unless those directories are located on the root file system).

This command will print the file system operations executed. This output may be suppressed by passing `--quiet`.

Note that this operation creates only the symlinks suggested in the "`[Install]`" section of the unit files. While this command is

the recommended way to manipulate the unit configuration directory, the administrator is free to make additional changes manually by placing or removing symlinks below this directory. This is particularly useful to create configurations that deviate from the suggested default installation. In this case, the administrator must make sure to invoke **daemon-reload** manually as necessary, in order to ensure the changes are taken into account.

Enabling units should not be confused with starting (activating) units, as done by the **start** command. Enabling and starting units is orthogonal: units may be enabled without being started and started without being enabled. Enabling simply hooks the unit into various suggested places (for example, so that the unit is automatically started on boot or when a particular kind of hardware is plugged in). Starting actually spawns the daemon process (in case of service units), or binds the socket (in case of socket units), and so on.

Depending on whether `--system`, `--user`, `--runtime`, or `--global` is specified, this enables the unit for the system, for the calling user only, for only this boot of the system, or for all future logins of all users. Note that in the last case, no systemd daemon configuration is reloaded.

Using **enable** on masked units is not supported and results in an error.

disable ***UNIT*****...**

Disables one or more units. This removes all symlinks to the unit files backing the specified units from the unit configuration directory, and hence undoes any changes made by **enable** or **link**. Note that this removes *all* symlinks to matching unit files, including manually created symlinks, and not just those actually created by **enable** or **link**. Note that while **disable** undoes the effect of **enable**, the two commands are otherwise not symmetric, as **disable** may remove more symlinks than a prior **enable** invocation of the same unit created.

This command expects valid unit names only, it does not accept paths to unit files.

In addition to the units specified as arguments, all units are disabled that are listed in the `Also=` setting contained in the "`[Install]`" section of any of the unit files being operated on.

This command implicitly reloads the system manager configuration after completing the operation. Note that this command does not implicitly stop the units that are being disabled. If this is desired, either combine this command with the `--now` switch, or invoke the **stop** command with appropriate arguments later.

This command will print information about the file system operations (symlink removals) executed. This output may be suppressed by passing `--quiet`.

This command honors `--system`, `--user`, `--runtime` and `--global` in a similar way as **enable**.

reenable ***UNIT*****...**

Reenable one or more units, as specified on the command line. This is a combination of **disable** and **enable** and is useful to reset the symlinks a unit file is enabled with to the defaults configured in its "`[Install]`" section. This command expects a unit name only, it does not accept paths to unit files.

preset ***UNIT*****...**

Reset the enable/disable status one or more unit files, as specified on the command line, to the defaults configured in the preset policy files. This has the same effect as **disable** or **enable**, depending how the unit is listed in the preset files.

Use `--preset-mode=` to control whether units shall be enabled and disabled, or only enabled, or only disabled.

If the unit carries no install information, it will be silently ignored by this command. *`UNIT`* must be the real unit name, any alias names are ignored silently.

For more information on the preset policy format, see systemd.preset(5). For more information on the concept of presets, please consult the Preset document.

preset-all

Resets all installed unit files to the defaults configured in the preset policy file (see above).

Use `--preset-mode=` to control whether units shall be enabled and disabled, or only enabled, or only disabled.

is-enabled ***UNIT*****...**

Checks whether any of the specified unit files are enabled (as with **enable**). Returns an exit code of 0 if at least one is enabled, non-zero otherwise. Prints the current enable status (see table). To suppress this output, use `--quiet`. To show installation targets, use `--full`.

Table 1. is-enabled output

Name	Description	Exit Code
"`enabled`"	Enabled via `.wants/`, `.requires/` or `Alias=` symlinks (permanently in `/etc/systemd/system/`, or transiently in `/run/systemd/system/`).	0
"`enabled-runtime`"		
"`linked`"	Made available through one or more symlinks to the unit file (permanently in `/etc/systemd/system/` or transiently in `/run/systemd/system/`), even though the unit file might reside outside of the unit file search path.	> 0
"`linked-runtime`"		
"`masked`"	Completely disabled, so that any start operation on it fails (permanently in `/etc/systemd/system/` or	> 0

Name	Description	Exit Code
"masked-runtime"	transiently in /run/systemd/systemd/).	
"static"	The unit file is not enabled, and has no provisions for enabling in the "[Install]" unit file section.	0
"indirect"	The unit file itself is not enabled, but it has a non-empty Also= setting in the "[Install]" unit file section, listing other unit files that might be enabled, or it has an alias under a different name through a symlink that is not specified in Also=. For template unit file, an instance different than the one specified in DefaultInstance= is enabled.	0
"disabled"	The unit file is not enabled, but contains an "[Install]" section with installation instructions.	> 0
"generated"	The unit file was generated dynamically via a generator tool. See systemd.generator(7). Generated unit files may not be enabled, they are enabled implicitly by their generator.	0
"transient"	The unit file has been created dynamically with the runtime API. Transient units may not be enabled.	0
"bad"	The unit file is invalid or another error occurred. Note that **is-enabled** will not actually return this state, but print an error message instead. However the unit file listing printed by **list-unit-files** might show it.	> 0

mask *UNIT*...

Mask one or more units, as specified on the command line. This will link these unit files to /dev/null, making it impossible to start them. This is a stronger version of **disable**, since it prohibits all kinds of activation of the unit, including enablement and manual activation. Use this option with care. This honors the --runtime option to only mask temporarily until the next reboot of the system. The --now option may be used to ensure that the units are also stopped. This command expects valid unit names only, it does not accept unit file paths.

unmask *UNIT*...

Unmask one or more unit files, as specified on the command line. This will undo the effect of **mask**. This command expects valid unit names only, it does not accept unit file paths.

link *PATH*...

Link a unit file that is not in the unit file search paths into the unit file search path. This command expects an absolute path to a unit file. The effect of this may be undone with **disable**. The effect of this command is that a unit file is made available for commands such as **start**, even though it is not installed directly in the unit search path. The file system where the linked unit files are located must be accessible when systemd is started (e.g. anything underneath /home or /var is not allowed, unless those directories are located on the root file system).

revert *UNIT*...

Revert one or more unit files to their vendor versions. This command removes drop-in configuration files that modify the specified units, as well as any user-configured unit file that overrides a matching vendor supplied unit file. Specifically, for a unit "foo.service" the matching directories "foo.service.d/" with all their contained files are removed, both below the persistent and runtime configuration directories (i.e. below /etc/systemd/system and /run/systemd/system); if the unit file has a vendor-supplied version (i.e. a unit file located below /usr) any matching persistent or runtime unit file that overrides it is removed, too. Note that if a unit file has no vendor-supplied version (i.e. is only defined below /etc/systemd/system or /run/systemd/system, but not in a unit file stored below /usr), then it is not removed. Also, if a unit is masked, it is unmasked.

Effectively, this command may be used to undo all changes made with **systemctl edit**, **systemctl set-property** and **systemctl mask** and puts the original unit file with its settings back in effect.

add-wants *TARGET UNIT*..., **add-requires** *TARGET UNIT*...

Adds "Wants=" or "Requires=" dependencies, respectively, to the specified *TARGET* for one or more units.

This command honors --system, --user, --runtime and --global in a way similar to **enable**.

edit *UNIT*...

Edit a drop-in snippet or a whole replacement file if --full is specified, to extend or override the specified unit.

Depending on whether --system (the default), --user, or --global is specified, this command creates a drop-in file for each unit either for the system, for the calling user, or for all futures logins of all users. Then, the editor (see the "Environment" section below) is invoked on temporary files which will be written to the real location if the editor exits successfully.

If --full is specified, this will copy the original units instead of creating drop-in files.

If --force is specified and any units do not already exist, new unit files will be opened for editing.

If --runtime is specified, the changes will be made temporarily in /run and they will be lost on the next reboot.

If the temporary file is empty upon exit, the modification of the related unit is canceled.

After the units have been edited, systemd configuration is reloaded (in a way that is equivalent to **daemon-reload**).

Note that this command cannot be used to remotely edit units and that you cannot temporarily edit units which are in /etc, since they take precedence over /run.

get-default

Return the default target to boot into. This returns the target unit name default.target is aliased (symlinked) to.

set-default *TARGET*

Set the default target to boot into. This sets (symlinks) the `default.target` alias to the given target unit.

Machine Commands

list-machines [*PATTERN*…]

List the host and all running local containers with their state. If one or more *PATTERN*s are specified, only containers matching one of them are shown.

Job Commands

list-jobs [*PATTERN*…]

List jobs that are in progress. If one or more *PATTERN*s are specified, only jobs for units matching one of them are shown.

When combined with `--after` or `--before` the list is augmented with information on which other job each job is waiting for, and which other jobs are waiting for it, see above.

cancel *JOB*…

Cancel one or more jobs specified on the command line by their numeric job IDs. If no job ID is specified, cancel all pending jobs.

Environment Commands

show-environment

Dump the systemd manager environment block. This is the environment block that is passed to all processes the manager spawns. The environment block will be dumped in straight-forward form suitable for sourcing into most shells. If no special characters or whitespace is present in the variable values, no escaping is performed, and the assignments have the form "`VARIABLE=value`". If whitespace or characters which have special meaning to the shell are present, dollar-single-quote escaping is used, and assignments have the form "`VARIABLE=$'value'`". This syntax is known to be supported by bash(1), zsh(1), ksh(1), and busybox(1)'s ash(1), but not dash(1) or fish(1).

set-environment *VARIABLE=VALUE*…

Set one or more systemd manager environment variables, as specified on the command line.

unset-environment *VARIABLE*…

Unset one or more systemd manager environment variables. If only a variable name is specified, it will be removed regardless of its value. If a variable and a value are specified, the variable is only removed if it has the specified value.

import-environment [*VARIABLE*…]

Import all, one or more environment variables set on the client into the systemd manager environment block. If no arguments are passed, the entire environment block is imported. Otherwise, a list of one or more environment variable names should be passed, whose client-side values are then imported into the manager's environment block.

Manager Lifecycle Commands

daemon-reload

Reload the systemd manager configuration. This will rerun all generators (see systemd.generator(7)), reload all unit files, and recreate the entire dependency tree. While the daemon is being reloaded, all sockets systemd listens on behalf of user configuration will stay accessible.

This command should not be confused with the **reload** command.

daemon-reexec

Reexecute the systemd manager. This will serialize the manager state, reexecute the process and deserialize the state again. This command is of little use except for debugging and package upgrades. Sometimes, it might be helpful as a heavy-weight **daemon-reload**. While the daemon is being reexecuted, all sockets systemd listening on behalf of user configuration will stay accessible.

System Commands

is-system-running

Checks whether the system is operational. This returns success (exit code 0) when the system is fully up and running, specifically not in startup, shutdown or maintenance mode, and with no failed services. Failure is returned otherwise (exit code non-zero). In addition, the current state is printed in a short string to standard output, see the table below. Use `--quiet` to suppress this output.

Use `--wait` to wait until the boot process is completed before printing the current state and returning the appropriate error status. If `--wait` is in use, states `initializing` or `starting` will not be reported, instead the command will block until a later state (such as `running` or `degraded`) is reached.

Table 2. is-system-running output

Name	Description	Exit Code
`initializing`	Early bootup, before `basic.target` is reached or the `maintenance` state entered.	> 0
`starting`	Late bootup, before the job queue becomes idle for the first time, or one of the rescue targets are reached.	> 0
`running`	The system is fully operational.	0
`degraded`	The system is operational but one or more units failed.	> 0
`maintenance`	The rescue or emergency target is active.	> 0
`stopping`	The manager is shutting down.	> 0
`offline`	The manager is not running. Specifically, this is the operational state if an incompatible program is running as system manager (PID 1).	> 0
`unknown`	The operational state could not be determined, due to lack of resources or another error cause.	> 0

default

Enter default mode. This is equivalent to **systemctl isolate default.target**. This operation is blocking by default, use `--no-block` to request asynchronous behavior.

rescue

Enter rescue mode. This is equivalent to **systemctl isolate rescue.target**. This operation is blocking by default, use `--no-block` to request asynchronous behavior.

emergency

Enter emergency mode. This is equivalent to **systemctl isolate emergency.target**. This operation is blocking by default, use `--no-block` to request asynchronous behavior.

halt

Shut down and halt the system. This is mostly equivalent to **systemctl start halt.target --job-mode=replace-irreversibly --no-block**, but also prints a wall message to all users. This command is asynchronous; it will return after the halt operation is enqueued, without waiting for it to complete. Note that this operation will simply halt the OS kernel after shutting down, leaving the hardware powered on. Use **systemctl poweroff** for powering off the system (see below).

If combined with `--force`, shutdown of all running services is skipped, however all processes are killed and all file systems are unmounted or mounted read-only, immediately followed by the system halt. If `--force` is specified twice, the operation is immediately executed without terminating any processes or unmounting any file systems. This may result in data loss. Note that when `--force` is specified twice the halt operation is executed by **systemctl** itself, and the system manager is not contacted. This means the command should succeed even when the system manager has crashed.

poweroff

Shut down and power-off the system. This is mostly equivalent to **systemctl start poweroff.target --job-mode=replace-irreversibly --no-block**, but also prints a wall message to all users. This command is asynchronous; it will return after the power-off operation is enqueued, without waiting for it to complete.

If combined with `--force`, shutdown of all running services is skipped, however all processes are killed and all file systems are unmounted or mounted read-only, immediately followed by the powering off. If `--force` is specified twice, the operation is immediately executed without terminating any processes or unmounting any file systems. This may result in data loss. Note that when `--force` is specified twice the power-off operation is executed by **systemctl** itself, and the system manager is not contacted. This means the command should succeed even when the system manager has crashed.

reboot [*arg*]

Shut down and reboot the system. This is mostly equivalent to **systemctl start reboot.target --job-mode=replace-irreversibly --no-block**, but also prints a wall message to all users. This command is asynchronous; it will return after the reboot operation is enqueued, without waiting for it to complete.

If combined with `--force`, shutdown of all running services is skipped, however all processes are killed and all file systems are unmounted or mounted read-only, immediately followed by the reboot. If `--force` is specified twice, the operation is immediately executed without terminating any processes or unmounting any file systems. This may result in data loss. Note that when `--force` is specified twice the reboot operation is executed by **systemctl** itself, and the system manager is not contacted. This means the command should succeed even when the system manager has crashed.

If the optional argument *arg* is given, it will be passed as the optional argument to the reboot(2) system call. The value is architecture and firmware specific. As an example, "`recovery`" might be used to trigger system recovery, and "`fota`" might be used to trigger a “firmware over the air” update.

kexec

Shut down and reboot the system via **kexec**. This is equivalent to **systemctl start kexec.target --job-mode=replace-irreversibly --no-block**. This command is asynchronous; it will return after the reboot operation is enqueued, without waiting for it to complete.

If combined with `--force`, shutdown of all running services is skipped, however all processes are killed and all file systems are unmounted or mounted read-only, immediately followed by the reboot.

exit [*EXIT_CODE*]

Ask the service manager to quit. This is only supported for user service managers (i.e. in conjunction with the `--user` option) or in containers and is equivalent to **poweroff** otherwise. This command is asynchronous; it will return after the exit operation is enqueued, without waiting for it to complete.

The service manager will exit with the specified exit code, if *`EXIT_CODE`* is passed.

switch-root *`ROOT`* [*`INIT`*]

Switches to a different root directory and executes a new system manager process below it. This is intended for usage in initial RAM disks ("initrd"), and will transition from the initrd's system manager process (a.k.a. "init" process) to the main system manager process which is loaded from the actual host volume. This call takes two arguments: the directory that is to become the new root directory, and the path to the new system manager binary below it to execute as PID 1. If the latter is omitted or the empty string, a systemd binary will automatically be searched for and used as init. If the system manager path is omitted, equal to the empty string or identical to the path to the systemd binary, the state of the initrd's system manager process is passed to the main system manager, which allows later introspection of the state of the services involved in the initrd boot phase.

suspend

Suspend the system. This will trigger activation of the special target unit `suspend.target`. This command is asynchronous, and will return after the suspend operation is successfully enqueued. It will not wait for the suspend/resume cycle to complete.

hibernate

Hibernate the system. This will trigger activation of the special target unit `hibernate.target`. This command is asynchronous, and will return after the hibernation operation is successfully enqueued. It will not wait for the hibernate/thaw cycle to complete.

hybrid-sleep

Hibernate and suspend the system. This will trigger activation of the special target unit `hybrid-sleep.target`. This command is asynchronous, and will return after the hybrid sleep operation is successfully enqueued. It will not wait for the sleep/wake-up cycle to complete.

suspend-then-hibernate

Suspend the system and hibernate it after the delay specified in `systemd-sleep.conf`. This will trigger activation of the special target unit `suspend-then-hibernate.target`. This command is asynchronous, and will return after the hybrid sleep operation is successfully enqueued. It will not wait for the sleep/wake-up or hibernate/thaw cycle to complete.

Parameter Syntax

Unit commands listed above take either a single unit name (designated as *`UNIT`*), or multiple unit specifications (designated as *`PATTERN`*...). In the first case, the unit name with or without a suffix must be given. If the suffix is not specified (unit name is "abbreviated"), systemctl will append a suitable suffix, "`.service`" by default, and a type-specific suffix in case of commands which operate only on specific unit types. For example,

```
# systemctl start sshd
```

and

```
# systemctl start sshd.service
```

are equivalent, as are

```
# systemctl isolate default
```

and

```
# systemctl isolate default.target
```

Note that (absolute) paths to device nodes are automatically converted to device unit names, and other (absolute) paths to mount unit names.

```
# systemctl status /dev/sda
# systemctl status /home
```

are equivalent to:

```
# systemctl status dev-sda.device
# systemctl status home.mount
```

In the second case, shell-style globs will be matched against the primary names of all units currently in memory; literal unit names, with or without a suffix, will be treated as in the first case. This means that literal unit names always refer to exactly one unit, but globs may match zero units and this is not considered an error.

Glob patterns use fnmatch(3), so normal shell-style globbing rules are used, and "`*`", "`?`", "`[]`" may be used. See glob(7) for more details. The patterns are matched against the primary names of units currently in memory, and patterns which do not match anything are silently skipped. For example:

```
# systemctl stop sshd@*.service
```

will stop all `sshd@.service` instances. Note that alias names of units, and units that aren't in memory are not considered for glob expansion.

For unit file commands, the specified *`UNIT`* should be the name of the unit file (possibly abbreviated, see above), or the absolute

path to the unit file:

```
# systemctl enable foo.service
```

or

```
# systemctl link /path/to/foo.service
```

Exit status

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

systemctl uses the return codes defined by LSB, as defined in LSB 3.0.0.

Table 3. LSB return codes

Value	Description in LSB	Use in systemd
0	"program is running or service is OK"	unit is active
1	"program is dead and /var/run pid file exists"	unit *not* failed (used by **is-failed**)
2	"program is dead and /var/lock lock file exists"	unused
3	"program is not running"	unit is not active
4	"program or service status is unknown"	no such unit

The mapping of LSB service states to systemd unit states is imperfect, so it is better to not rely on those return values but to look for specific unit states and substates instead.

Environment

`$SYSTEMD_EDITOR`

> Editor to use when editing units; overrides `$EDITOR` and `$VISUAL`. If neither `$SYSTEMD_EDITOR` nor `$EDITOR` nor `$VISUAL` are present or if it is set to an empty string or if their execution failed, systemctl will try to execute well known editors in this order: editor(1), nano(1), vim(1), vi(1).

`$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
X
```

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).

See Also

systemd(1), journalctl(1), loginctl(1), machinectl(1), systemd.unit(5), systemd.resource-control(5), systemd.special(7), wall(1), systemd.preset(5), systemd.generator(7), glob(7)