Justin Ellingwood

How To Use Systemctl to Manage Systemd Services and Units

Posted February 1, 2015 1.3m SYSTEM TOOLS

Introduction

`Systemd` is an init system and system manager that is widely becoming the new standard for Linux machines. While there are considerable opinions about whether `systemd` is an improvement over the traditional `SysV` init systems it is replacing, the majority of distributions plan to adopt it or have already done so.

Due to its heavy adoption, familiarizing yourself with `systemd` is well worth the trouble, as it will make administering servers considerably easier. Learning about and utilizing the tools and daemons that comprise `systemd` will help you better appreciate the power, flexibility, and capabilities it provides, or at least help you to do your job with minimal hassle.

In this guide, we will be discussing the `systemctl` command, which is the central management tool for controlling the init system. We will cover how to manage services, check statuses, change system states, and work with the configuration files.

Please note that although `systemd` has become the default init system for many Linux distributions, it isn't implemented universally across all distros. As you go through this tutorial, if your terminal outputs the error `bash: systemctl is not installed` then it is likely that your machine has a different init system installed.

Service Management

The fundamental purpose of an init system is to initialize the components that must be started after the Linux kernel is booted (traditionally known as "userland" components). The init system is also used to manage services and daemons for the server at any point while the system is running. With that in mind, we will start with some simple service management operations.

In `systemd`, the target of most actions are "units", which are resources that `systemd` knows how to manage. Units are categorized by the type of resource they represent and they are defined with files known as unit files. The type of each unit can be inferred from the suffix on the end of the file.

For service management tasks, the target unit will be service units, which have unit files with a suffix of `.service`. However, for most service management commands, you can actually leave off the `.service` suffix, as `systemd` is smart enough to know that you probably want to operate on a service when using service management commands.

Starting and Stopping Services

To start a `systemd` service, executing instructions in the service's unit file, use the `start` command. If you are running as a non-root user, you will have to use `sudo` since this will affect the state of the operating system:

```
sudo systemctl start application.service
```

As we mentioned above, `systemd` knows to look for `*.service` files for service management commands, so the command could just as easily be typed like this:

```
sudo systemctl start application
```

Although you may use the above format for general administration, for clarity, we will use the `.service` suffix for the remainder of the commands to be explicit about the target we are operating on.

To stop a currently running service, you can use the `stop` command instead:

```
sudo systemctl stop application.service
```

Restarting and Reloading

To restart a running service, you can use the `restart` command:

```
sudo systemctl restart application.service
```

If the application in question is able to reload its configuration files (without restarting), you can issue the `reload` command to initiate that process:

```
sudo systemctl reload application.service
```

If you are unsure whether the service has the functionality to reload its configuration, you can issue the `reload-or-restart` command. This will reload the configuration in-place if available. Otherwise, it will restart the service so the new configuration is picked up:

```
sudo systemctl reload-or-restart application.service
```

Enabling and Disabling Services

The above commands are useful for starting or stopping commands during the current session. To tell `systemd` to start services automatically at boot, you must enable them.
To start a service at boot, use the `enable` command:

```
sudo systemctl enable application.service
```

This will create a symbolic link from the system's copy of the service file (usually in `/lib/systemd/system` or `/etc/systemd/system`) into the location on disk where `systemd` looks for autostart files (usually `/etc/systemd/system/some_target.target.wants`. We will go over what a target is later in this guide).

To disable the service from starting automatically, you can type:

```
sudo systemctl disable application.service
```

This will remove the symbolic link that indicated that the service should be started automatically.

Keep in mind that enabling a service does not start it in the current session. If you wish to start the service and enable it at boot, you will have to issue both the `start` and `enable` commands.

Checking the Status of Services

To check the status of a service on your system, you can use the `status` command:

```
systemctl status application.service
```

This will provide you with the service state, the cgroup hierarchy, and the first few log lines.

For instance, when checking the status of an Nginx server, you may see output like this:

```
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; vendor preset: disab
   Active: active (running) since Tue 2015-01-27 19:41:23 EST; 22h ago
 Main PID: 495 (nginx)
   CGroup: /system.slice/nginx.service
           ├─495 nginx: master process /usr/bin/nginx -g pid /run/nginx.pid; error_log
           └─496 nginx: worker process
Jan 27 19:41:23 desktop systemd[1]: Starting A high performance web server and a revers
Jan 27 19:41:23 desktop systemd[1]: Started A high performance web server and a reverse
```

This gives you a nice overview of the current status of the application, notifying you of any problems and any actions that may be required.

There are also methods for checking for specific states. For instance, to check to see if a unit is currently active (running), you can use the `is-active` command:

```
systemctl is-active application.service
```

This will return the current unit state, which is usually `active` or `inactive`. The exit code will be "0" if it is active, making the result simpler to parse programmatically.
To see if the unit is enabled, you can use the `is-enabled` command:

```
systemctl is-enabled application.service
```

This will output whether the service is `enabled` or `disabled` and will again set the exit code to "0" or "1" depending on the answer to the command question.

A third check is whether the unit is in a failed state. This indicates that there was a problem starting the unit in question:

```
systemctl is-failed application.service
```

This will return `active` if it is running properly or `failed` if an error occurred. If the unit was intentionally stopped, it may return `unknown` or `inactive`. An exit status of "0" indicates that a failure occurred and an exit status of "1" indicates any other status.

System State Overview

The commands so far have been useful for managing single services, but they are not very helpful for exploring the current state of the system. There are a number of `systemctl` commands that provide this information.

Listing Current Units

To see a list of all of the active units that `systemd` knows about, we can use the `list-units` command:

```
systemctl list-units
```

This will show you a list of all of the units that `systemd` currently has active on the system. The output will look something like this:

```
UNIT                                     LOAD   ACTIVE SUB     DESCRIPTION
atd.service                              loaded active running ATD daemon
avahi-daemon.service                     loaded active running Avahi mDNS/DNS-SD Stack
dbus.service                             loaded active running D-Bus System Message Bu
dcron.service                            loaded active running Periodic Command Schedu
dkms.service                             loaded active exited  Dynamic Kernel Modules
getty@tty1.service                       loaded active running Getty on tty1
. . .
```

The output has the following columns:

- **UNIT**: The `systemd` unit name
- **LOAD**: Whether the unit's configuration has been parsed by `systemd`. The configuration of loaded units is kept in memory.
- **ACTIVE**: A summary state about whether the unit is active. This is usually a fairly basic way to tell if the unit has started successfully or not.
- **SUB**: This is a lower-level state that indicates more detailed information about the unit. This often varies by unit type, state, and the actual method in which the unit runs.
- **DESCRIPTION**: A short textual description of what the unit is/does.

Since the `list-units` command shows only active units by default, all of the entries above will show "loaded" in the LOAD column and "active" in the ACTIVE column. This display is actually the default behavior of `systemctl` when called without additional commands, so you will see the same thing if you call `systemctl` with no arguments:

```
systemctl
```

We can tell `systemctl` to output different information by adding additional flags. For instance, to see all of the units that `systemd` has loaded (or attempted to load), regardless of whether they are currently active, you can use the `--all` flag, like this:

```
systemctl list-units --all
```

This will show any unit that `systemd` loaded or attempted to load, regardless of its current state on the system. Some units become inactive after running, and some units that `systemd` attempted to load may have not been found on disk.

You can use other flags to filter these results. For example, we can use the `--state=` flag to indicate the LOAD, ACTIVE, or SUB states that we wish to see. You will have to keep the `--all` flag so that `systemctl` allows non-active units to be displayed:

```
systemctl list-units --all --state=inactive
```

Another common filter is the `--type=` filter. We can tell `systemctl` to only display units of the type we are interested in. For example, to see only active service units, we can use:

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

Listing All Unit Files

The `list-units` command only displays units that `systemd` has attempted to parse and load into memory. Since `systemd` will only read units that it thinks it needs, this will not necessarily include all of the available units on the system. To see *every* available unit file within the `systemd` paths, including those that `systemd` has not attempted to load, you can use the `list-unit-files` command instead:

```
systemctl list-unit-files
```

Units are representations of resources that `systemd` knows about. Since `systemd` has not necessarily read all of the unit definitions in this view, it only presents information about the files themselves. The output has two columns: the unit file and the state.

```
UNIT FILE                                   STATE
proc-sys-fs-binfmt_misc.automount           static
dev-hugepages.mount                         static
dev-mqueue.mount                            static
proc-fs-nfsd.mount                          static
proc-sys-fs-binfmt_misc.mount               static
sys-fs-fuse-connections.mount               static
sys-kernel-config.mount                     static
sys-kernel-debug.mount                      static
tmp.mount                                   static
var-lib-nfs-rpc_pipefs.mount                static
org.cups.cupsd.path                         enabled
. . .
```

The state will usually be "enabled", "disabled", "static", or "masked". In this context, static means that the unit file does not contain an "install" section, which is used to enable a unit. As such, these units cannot be enabled. Usually, this means that the unit performs a one-off action or is used only as a dependency of another unit and should not be run by itself.

We will cover what "masked" means momentarily.

Unit Management

So far, we have been working with services and displaying information about the unit and unit files that `systemd` knows about. However, we can find out more specific information about units using some additional commands.

Displaying a Unit File

To display the unit file that `systemd` has loaded into its system, you can use the `cat` command (this was added in `systemd` version 209). For instance, to see the unit file of the `atd` scheduling daemon, we could type:

```
systemctl cat atd.service
```

```
[Unit]
Description=ATD daemon
[Service]
Type=forking
ExecStart=/usr/bin/atd
[Install]
WantedBy=multi-user.target
```

The output is the unit file as known to the currently running `systemd` process. This can be important if you have modified unit files recently or if you are overriding certain options in a unit file fragment (we will cover this later).

Displaying Dependencies

To see a unit's dependency tree, you can use the `list-dependencies` command:

```
systemctl list-dependencies sshd.service
```

This will display a hierarchy mapping the dependencies that must be dealt with in order to start the unit in question. Dependencies, in this context, include those units that are either required by or wanted by the units above it.

```
sshd.service
├─system.slice
└─basic.target
  ├─microcode.service
  ├─rhel-autorelabel-mark.service
  ├─rhel-autorelabel.service
  ├─rhel-configure.service
  ├─rhel-dmesg.service
  ├─rhel-loadmodules.service
  ├─paths.target
  ├─slices.target
. . .
```

The recursive dependencies are only displayed for `.target` units, which indicate system states. To recursively list all dependencies, include the `--all` flag.

To show reverse dependencies (units that depend on the specified unit), you can add the `--reverse` flag to the command. Other flags that are useful are the `--before` and `--after` flags, which can be used to show units that depend on the specified unit starting before and after themselves, respectively.

Checking Unit Properties

To see the low-level properties of a unit, you can use the `show` command. This will display a list of properties that are set for the specified unit using a `key=value` format:

```
systemctl show sshd.service
```

```
Id=sshd.service
Names=sshd.service
Requires=basic.target
Wants=system.slice
WantedBy=multi-user.target
Conflicts=shutdown.target
Before=shutdown.target multi-user.target
After=syslog.target network.target auditd.service systemd-journald.socket basic.target
Description=OpenSSH server daemon
. . .
```

If you want to display a single property, you can pass the `-p` flag with the property name. For instance, to see the conflicts that the `sshd.service` unit has, you can type:

```
systemctl show sshd.service -p Conflicts
```

```
Conflicts=shutdown.target
```

Masking and Unmasking Units

We saw in the service management section how to stop or disable a service, but `systemd` also has the ability to mark a unit as *completely* unstartable, automatically or manually, by linking it to `/dev/null`. This is called masking the unit, and is possible with the `mask` command:

```
sudo systemctl mask nginx.service
```

This will prevent the Nginx service from being started, automatically or manually, for as long as it is masked.

If you check the `list-unit-files`, you will see the service is now listed as masked:

```
systemctl list-unit-files
```

```
. . .
kmod-static-nodes.service             static
ldconfig.service                      static
mandb.service                         static
messagebus.service                    static
nginx.service                         masked
quotaon.service                       static
rc-local.service                      static
rdisc.service                         disabled
rescue.service                        static
. . .
```

If you attempt to start the service, you will see a message like this:

```
sudo systemctl start nginx.service
```

```
Failed to start nginx.service: Unit nginx.service is masked.
```

To unmask a unit, making it available for use again, simply use the `unmask` command:

```
sudo systemctl unmask nginx.service
```

This will return the unit to its previous state, allowing it to be started or enabled.

Editing Unit Files

While the specific format for unit files is outside of the scope of this tutorial, `systemctl` provides built-in mechanisms for editing and modifying unit files if you need to make adjustments. This functionality was added in `systemd` version 218.

The `edit` command, by default, will open a unit file snippet for the unit in question:

```
sudo systemctl edit nginx.service
```

This will be a blank file that can be used to override or add directives to the unit definition. A directory will be created within the `/etc/systemd/system` directory which contains the name of the unit with `.d` appended. For instance, for the `nginx.service`, a directory called `nginx.service.d` will be created.

Within this directory, a snippet will be created called `override.conf`. When the unit is loaded, `systemd` will, in memory, merge the override snippet with the full unit file. The snippet's directives will take precedence over those found in the original unit file.

If you wish to edit the full unit file instead of creating a snippet, you can pass the `--full` flag:

```
sudo systemctl edit --full nginx.service
```

This will load the current unit file into the editor, where it can be modified. When the editor exits, the changed file will be written to `/etc/systemd/system`, which will take precedence over the system's unit definition (usually found somewhere in `/lib/systemd/system`).

To remove any additions you have made, either delete the unit's `.d` configuration directory or the modified service file from `/etc/systemd/system`. For instance, to remove a snippet, we could type:

```
sudo rm -r /etc/systemd/system/nginx.service.d
```

To remove a full modified unit file, we would type:

```
sudo rm /etc/systemd/system/nginx.service
```

After deleting the file or directory, you should reload the `systemd` process so that it no longer attempts to reference these files and reverts back to using the system copies. You can do this by typing:

```
sudo systemctl daemon-reload
```

Adjusting the System State (Runlevel) with Targets

Targets are special unit files that describe a system state or synchronization point. Like other units, the files that define targets can be identified by their suffix, which in this case is `.target`. Targets do not do much themselves, but are instead used to group other units together.

This can be used in order to bring the system to certain states, much like other init systems use runlevels. They are used as a reference for when certain functions are available, allowing you to specify the desired state instead of the individual units needed to produce that state.

For instance, there is a `swap.target` that is used to indicate that swap is ready for use. Units that are part of this process can sync with this target by indicating in their configuration that they are `WantedBy=` or `RequiredBy=` the `swap.target`. Units that require swap to be available can specify this condition using the `Wants=`, `Requires=`, and `After=` specifications to indicate the nature of their relationship.

Getting and Setting the Default Target

The `systemd` process has a default target that it uses when booting the system. Satisfying the cascade of dependencies from that single target will bring the system into the desired state. To find the default target for your system, type:

```
systemctl get-default
```

```
multi-user.target
```

If you wish to set a different default target, you can use the `set-default`. For instance, if you have a graphical desktop installed and you wish for the system to boot into that by default, you can change your default target accordingly:

```
sudo systemctl set-default graphical.target
```

Listing Available Targets

You can get a list of the available targets on your system by typing:

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

Unlike runlevels, multiple targets can be active at one time. An active target indicates that `systemd` has attempted to start all of the units tied to the target and has not tried to tear them down again. To see all of the active targets, type:

```
systemctl list-units --type=target
```

Isolating Targets

It is possible to start all of the units associated with a target and stop all units that are not part of the dependency tree. The command that we need to do this is called, appropriately, `isolate`. This is similar to changing the runlevel in other init systems.

For instance, if you are operating in a graphical environment with `graphical.target` active, you can shut down the graphical system and put the system into a multi-user command line state by isolating the `multi-user.target`. Since `graphical.target` depends on `multi-user.target` but not the other way around, all of the graphical units will be stopped.

You may wish to take a look at the dependencies of the target you are isolating before performing this procedure to ensure that you are not stopping vital services:

```
systemctl list-dependencies multi-user.target
```

When you are satisfied with the units that will be kept alive, you can isolate the target by typing:

```
sudo systemctl isolate multi-user.target
```

Using Shortcuts for Important Events

There are targets defined for important events like powering off or rebooting. However, `systemctl` also has some shortcuts that add a bit of additional functionality.

For instance, to put the system into rescue (single-user) mode, you can just use the `rescue` command instead of `isolate rescue.target`:

```
sudo systemctl rescue
```

This will provide the additional functionality of alerting all logged in users about the event.

To halt the system, you can use the `halt` command:

```
sudo systemctl halt
```

To initiate a full shutdown, you can use the `poweroff` command:

```
sudo systemctl poweroff
```

A restart can be started with the `reboot` command:

```
sudo systemctl reboot
```

These all alert logged in users that the event is occurring, something that simply running or isolating the target will not do. Note that most machines will link the shorter, more conventional commands for these operations so that they work properly with `systemd`.

For example, to reboot the system, you can usually type:

```
sudo reboot
```

Conclusion

By now, you should be familiar with some of the basic capabilities of the `systemctl` command that allow you to interact with and control your `systemd` instance. The `systemctl` utility will be your main point of interaction for service and system state management.

While `systemctl` operates mainly with the core `systemd` process, there are other components to the `systemd` ecosystem that are controlled by other utilities. Other capabilities, like log management and user sessions are handled by separate daemons and management utilities (`journald`/`journalctl` and `logind`/`loginctl` respectively). Taking time to become familiar with these other tools and daemons will make management an easier task.

Related Tutorials

How To Configure OpenLDAP and Perform Administrative LDAP Tasks

How To Change Account Passwords on an OpenLDAP Server

How To Use LDIF Files to Make Changes to an OpenLDAP System

How To Manage and Use LDAP Servers with OpenLDAP Utilities

How To Use Vundle to Manage Vim Plugins on a Linux VPS

19 Comments

Leave a comment...

Log In to Comment

thedude *February 2, 2015*

0

Thanks for this, very nice intro :)

There is a typo in

> To see if the unit is enabled, you can use the is-enabled command:
>
> ```
> systemctl is-active application.service
> ```

In the code block you mean `is-enabled`.

jellingwood MOD *February 2, 2015*

1

Oops! Must have missed that one. Thanks for letting me know!

leakybocks *February 17, 2015*

0

Very useful introduction to systemd. Next time I start a CentOS 7 droplet it'll come in handy!

Under the target section "Getting and Setting the Default Target" is "mulit-user.target" a typo? Should it say multi-user not mulit-user?

jellingwood MOD *February 17, 2015*

0

@leakybocks: Thanks for the feedback!

And good eye. I've fixed that up. Let me know if you see anything else!

kein.1945 *February 20, 2015*

0

What difference between `reboot` and `systemctl reboot` ?

jellingwood MOD *February 20, 2015*

0

@kein.1945: On most systems with `systemd`, the `reboot` command will actually be replaced by a symbolic link to the `systemctl` command, so effectively, they do the same thing. The only difference is that if you call it with `systemctl` it might write a message to any logged in users immediately prior to executing the reboot.

oviliz *December 30, 2015*

0

I was logged in with a different SSH user and haven't received any message when doing `systemctl reboot`.

antiquity *March 23, 2016*

0

Thank you for clear explanation.

nicolasliveris *March 30, 2016*

1

hi !
I have created a droplet with node and i want to create a service to start a node app on boot. However when i launch my service i have that:
'systemctl: command not found'

how to install the package systemctl ?

imewx *October 22, 2016*

0

Thank you so much!

bechesa *November 16, 2016*

0

Thanks for the tutorial

kelousami *February 16, 2017*

0

Thank you so much. Very clear!

rh *March 14, 2017*

0

Very useful ... ;but there does not seem to be a reference to:

```
systemctl daemon-reexec
```

to restart systemd

---

pauloporto *October 4, 2017*

0

Thank you so much Jellinwood. The Tuto ist very clear and cool. About the thema Systemd, do you have some Literary suggestion? what i mean is, more or extra deep information about. Thanks again!

---

jellingwood MOD *October 5, 2017*

0

@pauloporto If you'd like to get more in-depth information about systemd, a good place to go is Lennart Poettering's blog, the project's initial author and designer. He has some extensive essays on the internals and philosophy behind the project.

You can find a list of his posts here. The first post regarding systemd is called Rethinking PID 1. Afterwards, he has a series of posts on systemd for administrators, starting here and a series of posts on systemd for developers starting here. There are quite a few additional posts about key features, etc. I hope that helps!

---

ashokc *November 24, 2017*

0

Excellent article, Justin. Nice overview, compact & to the point. Thanks

---

Babblo *November 29, 2017*

0

Excellent info, thank you very much. Absolutely f*cked init system.

---

ale633 *December 10, 2017*

0

Very good tutorial on systemd and explanations appreciated. Thx!

---

jdp12383 *January 27, 2018*

0

Thanks Justin for such a nice tutorial. I was looking for beginner tutorial to understand systemctl and how it works to familiarize beginner like me.

I've two boards on device B1 & B2. B1 is hosting the web ui and browser can access it via B2. B2 has simple firewall service running which is routing packets to B1. Now I'm trying to solve a problem where nginx.conf on B1 requires ssl certificate files which are generated by a different process running on B2. B2 has the location /mnt/fromB1 mounted which is of B1. B2 process generates certificates and stores it at the mounted location so nginx on B1 can use it. This is because I don't want to maintain two copies of same certificates.

Now the problem I'm facing is when B1 boots up the very first time and nginx is ready to start, it can't find the certificates at the location and fail to launch. This is also preventing port 80 server instances not to start.

After reading your tutorial the solution I'm thinking is like I can add a service to start before nginx.service where I'll just create dummy certificate so to allow nginx to launch. Later when B2 finish initialization and the process is started it will overwrite my certificates. Though I don't know how to write a service to execute openssl commands using the .service files.

I'm familiar with CentOS and I could have modified the nginx.initscript and executed openssl commands there and then start nginx to solve this.

Appreciate any guidance in resolving this.