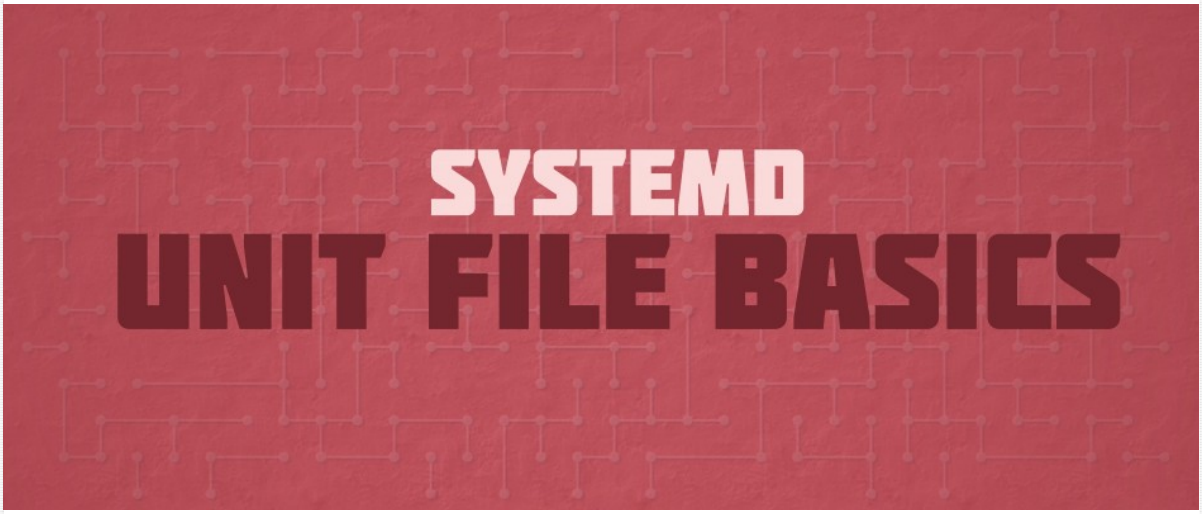
# systemd unit file basics




Welcome back to the systemd series, where we explore more about how this central part of your Fedora system works. This installment talks about *unit files*.

As a long-time Fedora user, I hadn't thought much about systemd actually worked until recently. systemd is compartmentalized so that components of your system can be more easily managed. systemd uses *unit files* to configure and manage system resources such as processes and your file system. Using these files you can wrangle systemd to manage your Fedora system the way the you want.

## Unit files: what are they?

The unit files on your system determine how systemd will start and run. Each corresponds to a single activity or component — or *unit* in systemd terms. Each unit file is a simple text file describing a unit, what it does, what needs to run before or afterward, and other details.

Unit files can be stored in a few different places on your system. systemd looks for

system unit files in this order:

1. /etc/systemd/system
2. /run/systemd/system
3. /usr/lib/systemd/system

Unit files in the earlier directories override later ones. This is a useful scheme, because it lets you make changes in the /etc directory, where configuration is expected. You should avoid making changes in /usr. Your system installs package data there that's not expected to change.

systemd can also run in a user context, and manage resources per user in addition to system-side. Unit files for user units are stored similarly in */etc/systemd/user, /run/systemd/user,* and */usr/lib/systemd/user.* The order of precedence works similarly.

You can examine some details about these unit files using the *systemctl* command. If you want to see a list of all the unit files installed on the system, run this command:

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

Each unit file contains options in the form *OptionName=value*, separated into sections marked like *[SectionName].* These options describe how the unit works, and how systemd deals with it.

There are numerous types of units systemd understands. The two most common for system owners to deal with are *service* units and *target* units. To list unit files on your system of each of these types, use the *systemctl* command:

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

## Service units

These are the units describe a process systemd can start and monitor. Service units are the most common units you'll use daily. They are controlled using *systemctl* as root:

```
sudo systemctl [command] NAME.service
```

Typical commands include:

- *start:* starts a systemd unit

- *stop*: attempts to “nicely” end a service
- *status*: provides detailed information on a service
- *restart*: restarts (stops and then starts) the specified service
- *enable*: hooks (links) a unit to various places, for instance to run at boot
- *disable*: unhooks (unlinks) a unit, so it is not activated

The following example is the *sshd.service* unit file. It allows systemd to control the *sshd* daemon that allows remote access to your system via SSH (Secure Shell).

```
[Unit]
Description=OpenSSH server daemon
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target sshd-keygen.service
Wants=sshd-keygen.service

[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

[Install]
WantedBy=multi-user.target
```

This unit features both some common unit file options, and service unit-specific options. Common options seen here include a description and where to find documentation. There is obviously more in this unit file than this article can explain. But as you can probably guess from the *ExecStart* option, this unit runs the *sshd* daemon when started.

## Service owned processes

One interesting feature of systemd is that it monitors processes it starts with service units. To find out what processes are being monitored, use the *systemctl status* command. For instance, here is output from that command checking the status of the *sshd.service* unit:

```
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor
preset: disabled)
   Active: active (running) since Tue 2015-10-20 14:51:24 EDT; 1 weeks 0 days
ago
   Docs: man:sshd(8)
         man:sshd_config(5)
   Main PID: 1090 (sshd)
   CGroup: /system.slice/sshd.service
           └─1090 /usr/sbin/sshd -D
```

We can see that the process started by this service has process ID (PID) 1090. This process will continue to be monitored by systemd.

Notice also that the service has placed this process in a specially named *control group*. The control group (or "cgroup") allows systemd to manage associated processes together. In future articles in the series, we'll explore how this feature allows you tune performance and resource limits.

Thanks to knowing the processes owned by this service, systemd can also help you manage errant services. Normally when you stop a service, you'll use the *systemctl* command as mentioned earlier. For example, to stop the web server service:

```
systemctl stop httpd.service
```

But what if the service doesn't respond or cooperate? In this case, systemctl has a built in *kill* switch:

```
systemctl kill httpd.service
```

## Target units

Target units are used to link and group other units together to describe a desired system state. Some of these units may be services. Others may be additional target units with their own groups of units.

Here's an example, the *multi-user.target* unit file.

```
[Unit]
Description=Multi-User System
Documentation=man:systemd.special(7)
Requires=basic.target
Conflicts=rescue.service rescue.target
After=basic.target rescue.service rescue.target
AllowIsolate=yes
```

Notice that this target unit file doesn't contain a command to execute. Instead, it functions as purely as a way to connect other units (in this case, mostly targets) together. In this case:

- The *multi-user.target* requires that *basic.target* run successfully when *multi-user.target* is run.
- If the *rescue.service* or *rescue.target* units run, they will cause this unit to stop, and vice versa.
- The *multi-user.target* unit starts after *basic.target* starts, and after

*rescue.service* and *rescue.target* are run, if those are started.

Additionally this target unit includes the *AllowIsolate* option. This option allows your system to treat the *multi-user.target* unit as a boot target, using the *systemctl isolate* command.

Target units group other units together not just with their unit file contents. A target can also have a *.wants* directory that links to units which will be started along with the target. For instance, the */usr/lib/systemd/system/multi-user.target* file has an associated folder */usr/lib/systemd/system/multi-user.target.wants*. This directory contains links to units (not just services but other targets as well) that will run when this target is run. Each of those may have its own dependencies, or unit file options like Requires above, as well.

## Helpful references

For a deep dive on common Unit section options you can read the man page for units:

```
man systemd.unit
```

There are also specific documentation files for service and target files. You can read them with these commands:

```
man systemd.service
man systemd.target
```

If you're looking for more information on process and control group management in systemd, refer to this helpful blog entry.



**Bryan Sutherland**

**October 28, 2015**

**For System Administrators, For Users**

**Previous post**

**Next post**

## 4 Comments

ADD YOURS

**Zbigniew Jędrzejewski-Szmek**
October 28, 2015 at 10:32

It's not necessary, or recommended, to ever use systemctl kill to stop services. systemctl stop has a timeout of its own, and after this timeout is exceeded, it will proceed to kill -SIGKILL all processes of the service. Using systemctl kill also does not properly stop dependent processes beforehand.

> treat the multi-user.target unit as a boot target, using the systemctl isolate command
> I think it would better to say '… using the systemctl set-default command'.

**ifoolb**
October 31, 2015 at 07:11

How to write log to systemd journal, use syslog.h?

**Zbigniew Jędrzejewski-Szmek**
October 31, 2015 at 09:47

That's one way. Using syslog() you can write simple messages.

You can also use sd_journal_print() to send structured log messages with additional fields.

**Tal**
November 14, 2015 at 12:26

I often find that with the emergence of any new sufficiently complex technology that I need to figure out, the initial phase of wrapping my head around the ideas behind the technology is the hardest part. You've made this very easy with your systemd articles. I'm currently going through them one by one.

Thanks Bryan.

PS:

Reread the first sentence of the "Service units" section
Search this article for the words "as purely as a way"

## Leave a Reply

Your email address will not be published.

Name

Email

Website

Post Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.

**SUBSCRIBE TO FEDORA MAGAZINE**

Search form

Subscribe with RSS

or

Enter your email address below to receive notifications of new posts by email.

Email Address

Email Address

Subscribe