LINUX VOICE TUTORIAL

LINUX 101: GET THE MOST OUT OF SYSTEMD

MIKE SAUNDERS

It's mightily controversial – but Systemd is here to stay. Learn how to use its features, and (maybe) learn to love it too…

WHY DO THIS?

- Understand the big changes in modern distros.
- See how Systemd replaces SysVinit.
- Get to grips with units and the new journal.

Hate mail, personal insults, death threats – Lennart Poettering, the author of Systemd, is used to receiving these. The Red Hat employee recently ranted on Google+ about the nature of the FOSS community (**http://tinyurl.com/poorlennart**), lamenting that it's "quite a sick place to be in". In particular, he points to Linus Torvalds's highly acerbic mailing list posts, and accuses the kernel head honcho of setting the tone of online discussion, making personal attacks and derogatory comments the norm.

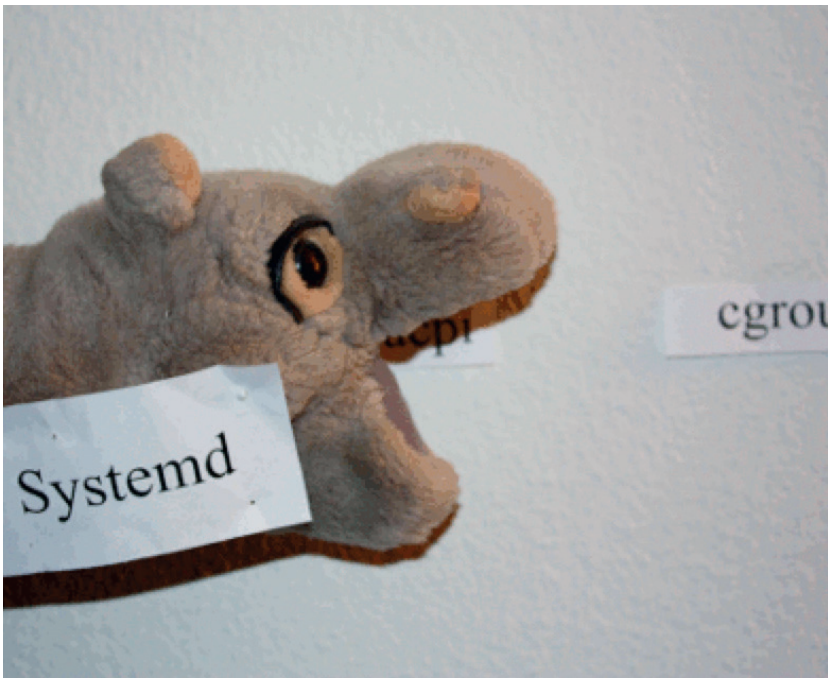
But why has Poettering received so much hate? Why does a man who simply develops open source software have to tolerate this amount of anger? Well, the answer lies in the importance of his software. Systemd is the first thing launched by the Linux kernel on most distributions now, and it serves many roles. It starts system services, handles logins, executes tasks at specified intervals, and much more. It's growing all the time, and becoming something of a "base system" for Linux – providing all the plumbing tools needed to boot and maintain a distro.

Now, Systemd is controversial for various reasons: it eschews some established Unix conventions, such as plain text log files. It's seen as a "monolithic" project trying to take over everything else. And it's a major change to the underpinnings of our OS. Yet almost every major distribution has adopted it (or is about to), so it's here to stay. And there are benefits: faster booting, easier management of services that depend on one another, and powerful and secure logging facilities too.

So in this tutorial we'll explore Systemd's features, and show you how to get the most out of them. Even if you're not a fan of the software right now, hopefully at least you'll feel more comfortable with it by the end.

1 BOOTING AND SERVICES

Almost every major distro has either adopted Systemd, or will do so in the next release (Debian and Ubuntu). In this tutorial we're using a pre-release of Fedora 21 – a distro that has been a great testing ground for Systemd – but the commands and notes should be the same regardless of your distro. That's one of the plus points of Systemd: it obviates many of the tiny, niggling differences between distros.



This tongue-in-cheek animation at **http://tinyurl.com/m2e7mv8** portrays Systemd as a rabid animal eating everything in its path. Most critics haven't been so fluffy.

In a terminal, enter **ps ax | grep systemd** and look at the first line. The **1** means that it's process ID 1, ie the first thing launched by the Linux kernel. So, once the kernel has done its work detecting hardware and organising memory, it launches the **/usr/lib/systemd/systemd** executable, which then launches other programs in turn. (In pre-Systemd days, the kernel would launch **/sbin/init**, which would then launch various other essential boot scripts, in a system known as SysVinit.)

Taking control

Central to Systemd is the concept of units. These are configuration files with information about services (programs running in the background), devices, mount points, timers and other aspects of the operating system. One of Systemd's goals is to ease and simplify the interaction between these, so if you have a certain program that needs to start when a certain mount point is created when a certain device gets plugged in, it should be considerably easier to make all this work. (In pre-Systemd days, hacking all this together with scripts could get very ugly.) To list all units on your Linux installation, enter:

Timer units: replacing Cron

Beyond system initialisation and service management, Systemd has its fingers in various other pies too. Notably, it can perform the job of **cron**, arguably with more flexibility (and an easier to read syntax). **Cron** is the program that performs jobs at regular intervals – such as cleaning up temporary files, refreshing caches and so forth.

If you look inside the **/usr/lib/systemd/system** directory again, you'll see that various **.timer** files are provided. Have a look at some of them with **less**, and you'll note that they follow a similar structure to the **.service** and **.target** files. The difference, however, lies in the **[Timer]** section. Consider this example:

```
[Timer]
OnBootSec=1h
OnUnitActiveSec=1w
```

Here, the **OnBootSec** option tells Systemd to activate the unit 1 hour after the system has booted. Then the second option means: activate the unit once a week after that. There's a huge amount of flexibility in the times that you can set – enter **man systemd.time** for a full list.

By default, Systemd's accuracy for timing is one minute. In other words, it will activate the unit within a minute of the time you specify, but not necessarily to the exact second. This is done for power management reasons, but if you need a timer to be executed without any delay, right down to the microsecond, you can add this line:

```
AccuracySec=1us
```

Also, the **WakeSystem** option (which can be set to true or false) defines whether or not the timer should wake up the machine if it's in suspend mode.

LV PRO TIP

By default, systemctl assumes that you're referring to services when issuing commands, so you can omit the **.service** bit in most cases. For instance, instead of entering **systemctl status gdm.service** you can just use **systemctl status gdm**. The same applies to stopping and starting services.

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

Now, **systemctl** is the main tool for interacting with Systemd, and it has many options. Here, in the unit list, you'll notice that there's some formatting: enabled units are shown in green, and disabled are shown in red. Units marked as "static" can't be started directly – they're dependencies of other units. To narrow down the list to just services, use:

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

Note that "enabled" doesn't necessarily mean that a service is running; just that it can be turned on. To get information about a specific service, for instance GDM (the Gnome Display Manager), enter:

```
systemctl status gdm.service
```

This provides lots of useful information: a human-readable description of the service, the location of the unit configuration file, when it was started, its PID, and the CGroups to which it belongs (these limit resource consumption for groups of processes).

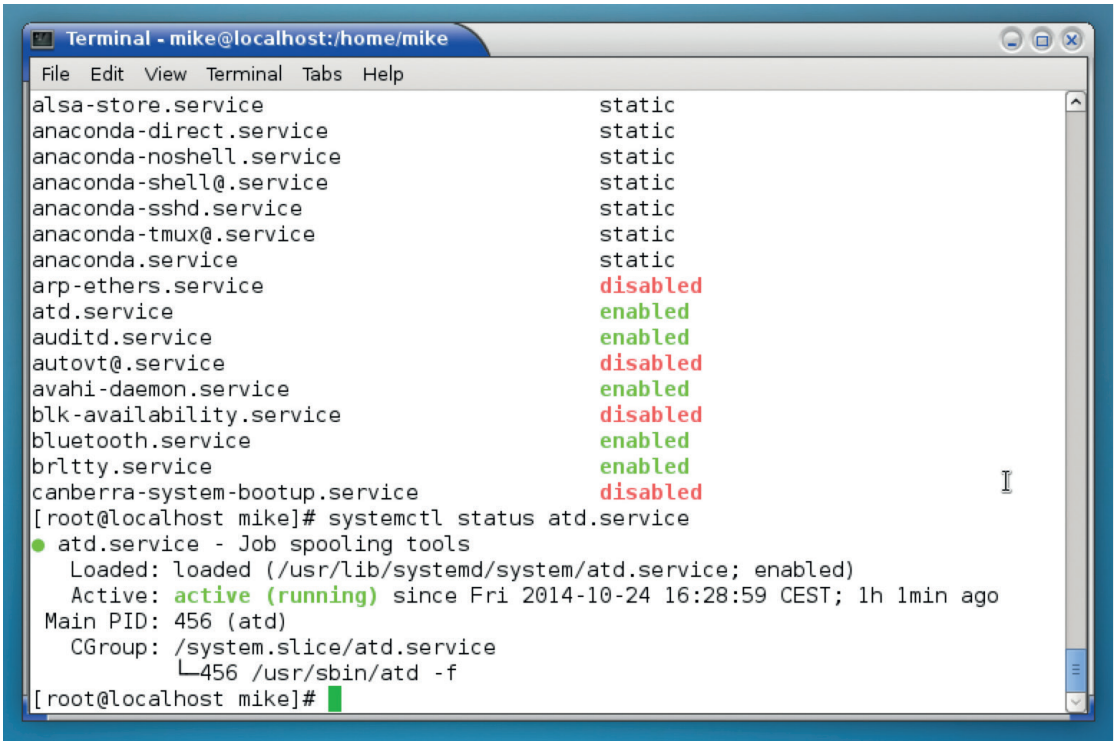
If you look at the unit config file in **/usr/lib/systemd/system/gdm.service**, you'll see various options, including the binary to be started (ExecStart), what it conflicts with (ie which units can't be active at the same time), and what needs to be started before this unit can be activated (the "After" line). Some units have additional dependency options, such as "Requires" (mandatory dependencies) and "Wants" (optional).

"Systemd eschews some established Unix conventions, such as plain text log files."

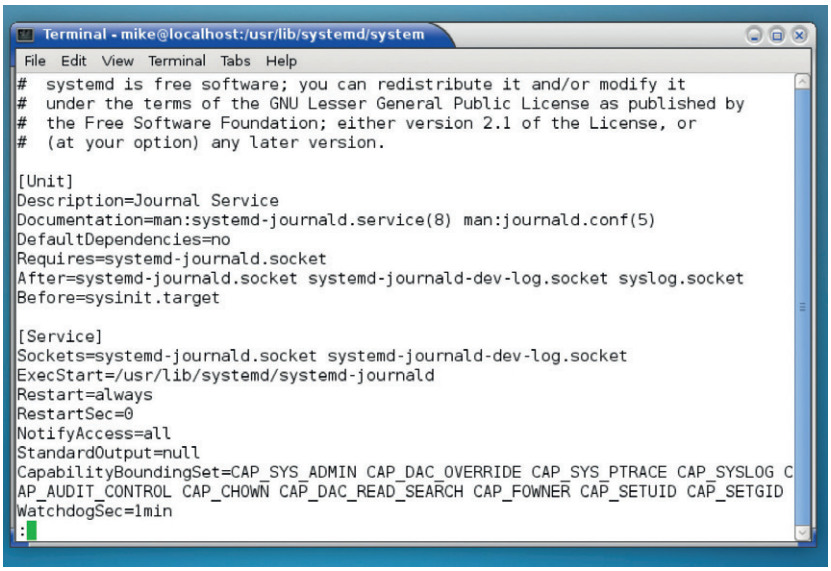
Another interesting option here is:

```
Alias=display-manager.service
```

When you activate **gdm.service**, you will also be able to view its status using **systemctl status display-manager.service**. This is useful when you know there's a display manager running, and you want



Use **systemctl status**, followed by a unit name, to see what's going on with a service.



The unit configuration files might look foreign compared to traditional scripts, but they're not hard to grasp.

to do something with it, but you don't care whether it's GDM, KDM, XDM or any of the others.

Target locked

If you enter **ls** in the **/usr/lib/systemd/system** directory, you'll also see various files that end in **.target**. A target is a way of grouping units together so that they're started at the same time. For instance, in most Unix-like OSes there's a state of the system called "multi-user", which means that the system has booted correctly, background services are running, and it's ready for one or more users to log in and do their work – at least, in text mode. (Other states include single-user, for doing administration work, or reboot, for when the machine is shutting down.)

If you look inside **multi-user.target**, you may be expecting to see a list of units that should be active in this state. But you'll notice that the file is pretty bare – instead, individual services make themselves dependencies of the target via the **WantedBy** option. So if you look inside **avahi-daemon.service**, **NetworkManager.service** and many other **.service** files, you'll see this line in the Install section:

```
WantedBy=multi-user.target
```

So, switching to the multi-user target will enable those units that contain the above line. Other targets are available (such as **emergency.target** for an emergency shell, or **halt.target** for when the machine shuts down), and you can easily switch between them like so:

```
systemctl isolate emergency.target
```

In many ways, these are like SysVinit runlevels, with text-mode **multi-user.target** being runlevel 3, **graphical.target** being runlevel 5, **reboot.target** being runlevel 6, and so forth.

Up and down

Now, you might be pondering: we've got this far, and yet we haven't even looked at stopping and starting services yet! But there's a reason for this. Systemd can look like a complicated beast from the outside, so it's good to have an overview of how it works before you start interacting with it. The actual commands for managing services are very simple:

```
systemctl stop cups.service
systemctl start cups.service
```

(If a unit has been disabled, you can first enable it with **systemctl enable** followed by the unit name. This places a symbolic link for the unit in the **.wants** directory of the current target, in the **/etc/systemd/system** folder.)

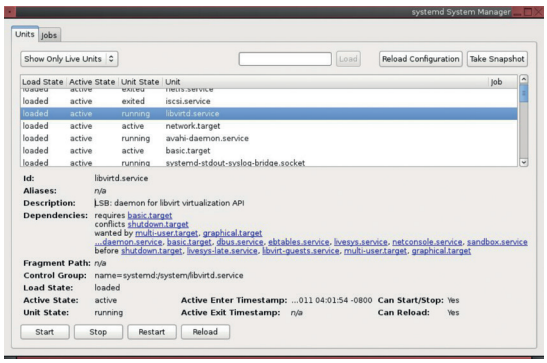
Two more useful commands are **systemctl restart** and **systemctl reload**, followed by unit names. The second asks the unit to reload its configuration file. Systemd is – for the most part – very well documented, so look at the manual page (**man systemctl**) for details on every command.

LV PRO TIP

A simple way to filter and manipulate the journal using regular Unix plain text tools is to use redirection. **journalctl -b > log.txt** will place all messages from the current boot in **log.txt**, so you can **sed** and **grep** to your heart's content.

2 LOG FILES: SAY HELLO TO JOURNALD

The second major component of Systemd is the journal. This is a logging system, similar to syslog, but with some major differences. And if you're a fan of the Unix way, prepare for your blood to boil: it's a binary log, so you can't just parse it using your regular command line text tools. This design decision regularly whips up heated debates on the net, but it has some benefits too. For instance, logs can be more structured, with better metadata, so it's easier to filter out information based on executable name, PID, time and so forth.



A Systemd GUI exists, although it hasn't been actively worked on for a couple of years.

To view the journal in its entirety, enter:

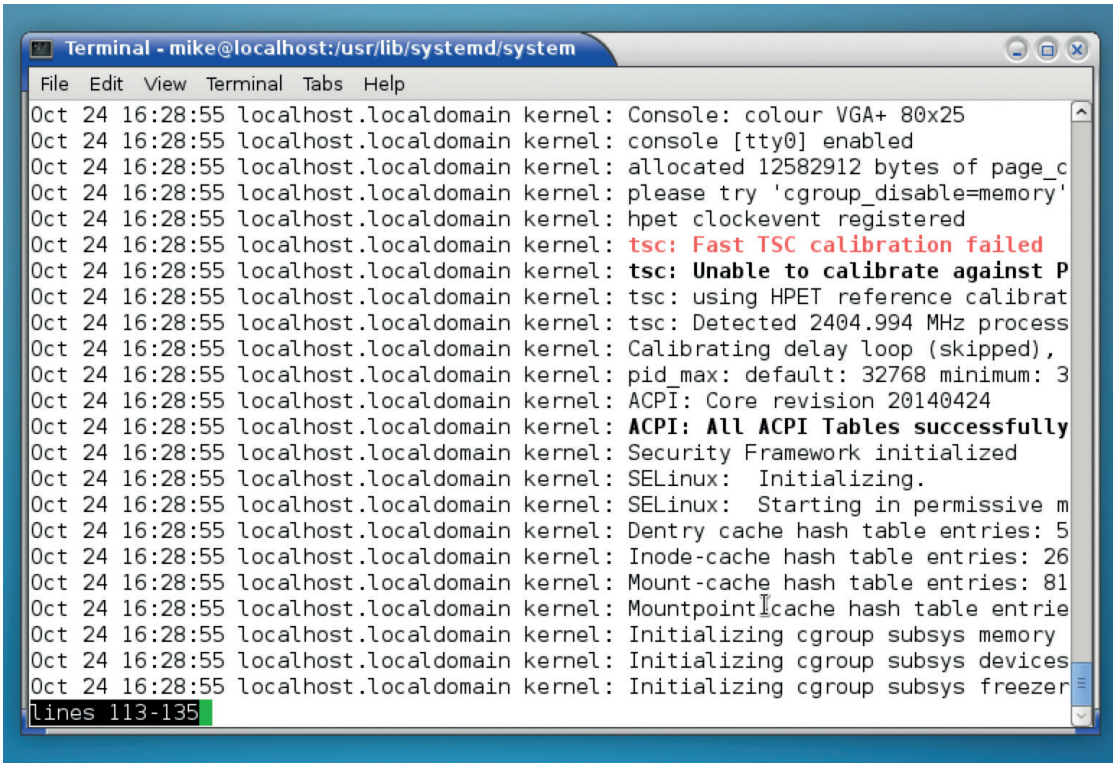
```
journalctl
```

As with many other Systemd commands, this pipes the output into the **less** program, so you can scroll down by hitting space, use **/** (forward slash) to search, and other familiar keybindings. You'll also notice a sprinkling of colour here too, with warnings and failure messages in red.

That's a lot of information; to narrow it down to the current boot, use:

```
journalctl -b
```

And here's where the Systemd journal starts to shine. Do you want to see all messages from the



Binary logging isn't popular, but the journal has some benefits, like very easy filtering of information.

previous boot? Try **journalctl -b -1**. Or the one before that? Replace **-1** with **-2**. How about something very specific, like all messages from 24 October 2014, 16:38 onwards?"

```
journalctl -b --since="2014-10-24 16:38"
```

Even if you deplore binary logs, that's still a useful feature, and for many admins it's much easier than constructing a similar filter from regular expressions.

So we've narrowed down the log to specific times, but what about specific programs? For units, try this:

```
journalctl -u gdm.service
```

(Note: that's a good way to see the log generated by the X server.) Or how about a specific PID?

```
journalctl _PID=890
```

You can even request to just see messages from a certain executable:

```
journalctl /usr/bin/pulseaudio
```

If you want to narrow down to messages of a certain priority, use the **-p** option. With 0 this will only show emergency messages (ie it's time to start praying to **$DEITY**), whereas 7 will show absolutely everything, including debugging messages. See the manual page (**man journalctl**) for more details on the priority levels.

It's worth noting that you can combine options as well, so to only show messages from the GDM service of priority level 3 (or lower) from the current boot, use:

```
journalctl -u gdm.service -p 3 -b
```

Finally, if you just want to have a terminal window open, constantly updating with the latest journal entries, as you'd have with the tail command in pre-Systemd installations, just enter **journalctl -f**.

Miked Saundersd has a PID of -1, divides by zero in his sleep, and knows how to sew on a button.

LV PRO TIP

We've been mostly poking around inside **/usr/lib/systemd/system** in this tutorial, but you may have noticed similar files inside **/etc/systemd/system** as well. What's the difference? Well, the latter takes precedence, so if you have two unit files with the same names in both locations, the one in **/etc/systemd/system** will be used. Generally, the former directory is where installed packages place their units, while the latter is for units created by root.

Life without Systemd?

If you simply, absolutely can't get on with Systemd, you still have a few choices among the major distributions. Most notably, Slackware, the longest-running distro, hasn't made the switch yet – but its lead developer hasn't ruled it out for the future. A few small-name distros are also holding out with SysVinit as well.

But how long will this last? Gnome is becoming increasingly dependent on Systemd, and the other major desktop environments could follow suit. This is a cause of consternation in the BSD communities, as Systemd is heavily tied to Linux kernel features, so the desktops are becoming less portable, in a way. A half-way-house solution might arrive in the form of Uselessd (**http://uselessd.darknedgy.net**), which is a stripped-down version of Systemd that purely focuses on launching and supervising processes, without consuming the whole base system.

If you don't like Sysytemd, try Gentoo, which has it as a choice of init system, but doesn't force it on its users.