Systemctl

Start/Stop/Restart

The .service extention is optional:

```
systemctl start application[.service]
systemctl stop application[.service]
systemctl restart application[.service]
```

Reload

In case the unit configuration file is changed reload it:

```
systemctl reload application.service
```

Enabling and Disabling

```
systemctl enable application[.service]
systemctl disable application[.service]
```

Checking the Status

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

```
systemctl status application[.service]
systemctl is-active application[.service]
systemctl is-enabled application[.service]
systemctl is-failed application[.service]
```

System State Overview

This will show you a list of all of the units that systemd currently has active on the system. 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.

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

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
systemctl list-units --all --state=inactive
systemctl list-units --all --state=active
systemctl list-units --all --type=service
systemctl list-units --all --type=timer
```

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
```

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.

Displaying Dependencies

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

```
systemctl list-dependencies cron.service
```

To see a unit’s reverse dependency tree:

```
systemctl list-dependencies --reverse cron.service
```

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.

System State (Runlevel) with Targets

Getting and Setting the Default Target

```
systemctl get-default
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
```

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

Analyze

First let's find out the actual boot time of the machine by running the command with no arguments, like so:

```
systemd-analyze
---------------
Startup finished in 1min 12.678s (userspace)
graphical.target reached after 4.651s in userspace
```

If that time is too long for you, how do you find out what's to blame? I'm glad you asked. Issue the command:

```
systemd-analyze blame
---------------------
          3.138s networking.service
          1.205s ifupdown-pre.service
           129ms exim4.service
            89ms systemd-udev-trigger.service
            50ms systemd-update-utmp.service
            44ms systemd-logind.service
            28ms systemd-journald.service
            19ms systemd-sysctl.service
            18ms systemd-sysusers.service
            17ms systemd-udevd.service
```

```
            16ms systemd-journal-flush.service
            15ms systemd-tmpfiles-setup-dev.service
            13ms systemd-tmpfiles-setup.service
            12ms systemd-tmpfiles-clean.service
             7ms systemd-user-sessions.service
             6ms dev-mqueue.mount
             5ms systemd-update-utmp-runlevel.service
             5ms systemd-remount-fs.service
             5ms dev-hugepages.mount
             3ms var-tmp.mount
             3ms tmp.mount
             2ms var-log.mount
```

You can also have the command print out the results in a chain of events style with the command:

```
systemd-analyze critical-chain
------------------------------
The time after the unit is active or started is printed after the "@"
character.
The time the unit takes to start is printed after the "+" character.

graphical.target @4.651s
└─multi-user.target @4.651s
  └─exim4.service @4.522s +129ms
    └─network-online.target @4.521s
      └─network.target @4.520s
        └─networking.service @1.382s +3.138s
          └─ifupdown-pre.service @175ms +1.205s
            └─systemd-udev-trigger.service @84ms +89ms
              └─systemd-journald.socket @83ms
                └─system.slice @72ms
                  └─-.slice @72ms
```

Check startup sequence

Systemd has the possibility to graphically show the startup sequence of all the services. This can be used to validate the timing of the service during boot.

```
systemd-analyze plot > startup_order.svg
systemd-analyze --host=root@192.168.178.61 plot > startup_order.svg
```