systemd

From the **project web page (https://systemd.io/)**:

> *systemd* is a suite of basic building blocks for a Linux system. It provides a system and service manager that runs as PID 1 and starts the rest of the system. *systemd* provides aggressive parallelization capabilities, uses socket and **D-Bus** activation for starting services, offers on-demand starting of daemons, keeps track of processes using Linux **control groups**, maintains mount and automount points, and implements an elaborate transactional dependency-based service control logic. *systemd* supports SysV and LSB init scripts and works as a replacement for sysvinit. Other parts include a logging daemon, utilities to control basic system configuration like the hostname, date, locale, maintain a list of logged-in users and running containers and virtual machines, system accounts, runtime directories and settings, and daemons to manage simple network configuration, network time synchronization, log forwarding, and name resolution.

Related articles

- **systemd/User**
- **systemd/Timers**
- **systemd/Journal**
- **systemd/Sandboxing**
- **systemd/FAQ**
- **init**
- **udev**
- **Improving performance/Boot process**
- **Allow users to shutdown**

Historically, what systemd calls "service" was named **daemon**: any program that runs as a "background" process (without a terminal or user interface), commonly waiting for events to occur and offering services. A good example is a web server that waits for a request to deliver a page, or an ssh server waiting for someone trying to log in. While these are full featured applications, there are daemons whose work is not that visible. Daemons are for tasks like writing messages into a log file (e.g. `syslog`, `metalog`) or keeping your system time accurate (e.g. **ntpd**). For more information see **`daemon(7) (https://man.archlinux.org/man/daemon.7)`**.

Note

For a detailed explanation of why Arch moved to *systemd*, see **this forum post (https://bbs.archlinux.org/viewtopic.php?pid=1149530#p1149530)**.

1 Basic systemctl usage

The main command used to introspect and control *systemd* is *systemctl*. Some of its uses are examining the system state and managing the system and services. See **`systemctl(1) (https://man.archlinux.org/man/systemctl.1)`** for more details.

Tip

- You can use all of the following *systemctl* commands with the `-H user@host` switch to control a *systemd* instance on a remote machine. This will use **SSH** to connect to the remote *systemd* instance.

- **Plasma** users can install `systemdgenie` (https://archlinux.org/packages/?name=systemdgenie) as a graphical frontend for *systemctl*.

1.1 Using units

Units commonly include, but are not limited to, services (*.service*), mount points (*.mount*), devices (*.device*) and sockets (*.socket*).

When using *systemctl*, you generally have to specify the complete name of the unit file, including its suffix, for example `sshd.socket`. There are however a few short forms when specifying the unit in the following *systemctl* commands:

- If you do not specify the suffix, systemctl will assume *.service*. For example, `netctl` and `netctl.service` are equivalent.
- Mount points will automatically be translated into the appropriate *.mount* unit. For example, specifying `/home` is equivalent to `home.mount`.
- Similar to mount points, devices are automatically translated into the appropriate *.device* unit, therefore specifying `/dev/sda2` is equivalent to `dev-sda2.device`.

See `systemd.unit(5)` (https://man.archlinux.org/man/systemd.unit.5) for details.

Note

Some unit names contain an `@` sign (e.g. `name@*string*.service`): this means that they are **instances (https://0pointer.net/blog/projects/instances.html)** of a *template* unit, whose actual file name does not contain the `*string*` part (e.g. `name@.service`). `*string*` is called the *instance identifier*, and is similar to an argument that is passed to the template unit when called with the *systemctl* command: in the unit file it will substitute the `%i` specifier. To be more accurate, *before* trying to instantiate the `name@.*suffix*` template unit, *systemd* will actually look for a unit with the exact `name@*string*.*suffix*` file name, although by convention such a "clash" happens rarely, i.e. most unit files containing an `@` sign are meant to be templates. Also, if a template unit is called without an instance identifier, it will generally fail (except with certain *systemctl* commands, like `cat`).

The commands in the below table operate on **system units** since `--system` is the implied default for *systemctl*. To instead operate on **user units** (for the *calling user*), use **systemctl --user** without root privileges. See also **systemd/User#Basic setup** to enable/disable user units for *all users*.

Tip

- Most commands also work if multiple units are specified, see `systemctl(1)` (https://man.archlinux.org/man/systemctl.1) for more information.
- The `--now` switch can be used in conjunction with `enable`, `disable`, and `mask` to respectively start, stop, or mask the unit *immediately* rather than after rebooting.
- A package may offer units for different purposes. If you just installed a package, `pacman -Qql *package* | grep -Fe .service -e .socket` can be used to check and find them.
- When available, enabling `*unit*.socket` instead of `*unit*.service` might be beneficial because the socket would start the service when necessary. See **#Socket activation** for

more details.

Action	Command	Note
Analyzing the system state		
Show system status	`systemctl status`	
List running units	`systemctl` or `systemctl list-units`	
List failed units	`systemctl --failed`	
List installed unit files[1]	`systemctl list-unit-files`	
Show process status for a PID	`systemctl status pid`	cgroup slice, memory and parent
Checking the unit status		
Show a manual page associated with a unit	`systemctl help unit`	as supported by the unit
Status of a unit	`systemctl status unit`	including whether it is running or not
Check whether a unit is enabled	`systemctl is-enabled unit`	
Starting, restarting, reloading a unit		
Start a unit immediately	`systemctl start unit` as root	
Stop a unit immediately	`systemctl stop unit` as root	
Restart a unit	`systemctl restart unit` as root	
Reload a unit and its configuration	`systemctl reload unit` as root	
Reload systemd manager configuration[2]	`systemctl daemon-reload` as root	scan for new or changed units
Enabling a unit		
Enable a unit to start automatically at boot	`systemctl enable unit` as root	
Enable a unit to start automatically at boot and **start** it immediately	`systemctl enable --now unit` as root	
Disable a unit to no longer start at boot	`systemctl disable unit` as root	
Reenable a unit[3]	`systemctl reenable unit` as root	i.e. disable and enable anew
Masking a unit		
Mask a unit to make it impossible to start[4]	`systemctl mask unit` as root	
Unmask a unit	`systemctl unmask unit` as root	

1. See [systemd.unit(5) § UNIT FILE LOAD PATH (https://man.archlinux.org/man/systemd.unit.5#UNIT_FILE_LOAD_PATH)] for the directories where available unit files can be found.
2. This does not ask the changed units to reload their own configurations (see the **Reload** action).
3. For example, in case its `[Install]` section has changed since last enabling it.
4. Both manually and as a dependency, which makes masking dangerous. Check for existing masked units with:

```
$ systemctl list-unit-files --state=masked
```

1.2 Power management

polkit is necessary for power management as an unprivileged user. If you are in a local *systemd-logind* user session and no other session is active, the following commands will work without root privileges. If not (for example, because another user is logged into a tty), *systemd* will automatically ask you for the root password.

Action	Command
Shut down and reboot the system	`systemctl reboot`
Shut down and power-off the system	`systemctl poweroff`
Suspend the system	`systemctl suspend`
Put the system into hibernation (write RAM to disk)	`systemctl hibernate`
Put the system into hybrid-sleep state (also called suspend-to-both, it saves RAM to disk and then suspends)	`systemctl hybrid-sleep`
First suspend the system, then wake up after a configured time in order to just hibernate the system	`systemctl suspend-then-hibernate`
Perform a reboot of the userspace-only with a **#Soft reboot**.	`systemctl soft-reboot`

1.2.1 Soft reboot

Soft reboot is a special kind of a userspace-only reboot operation that does not involve the kernel. It is implemented by systemd-soft-reboot.service(8) (https://man.archlinux.org/man/systemd-soft-reboot.service.8) and can be invoked through `systemctl soft-reboot`. As with **kexec**, it skips firmware re-initialization, but additionally the system does not go through kernel initialization and **initramfs**, and unlocked **dm-crypt** devices remain attached.

When `/run/nextroot/` contains a valid root file system hierarchy (e.g. is the root mount of another distribution or another snapshot), *soft-reboot* would switch the system root into it, allowing for switching to another installation without losing states managed by kernel, e.g. **networking**.

Tip

`/run/nextroot/` is not necessarily a mount point or backed by physical device. For example, it can reside in the `/run/` tmpfs. *systemd* will turn `/run/nextroot/` automatically into a mount point on *soft-reboot*.

Note

Do not invoke `systemctl soft-reboot` after package updates that involved the kernel and initramfs.

2 Writing unit files

The syntax of *systemd*'s unit files (systemd.unit(5) (https://man.archlinux.org/man/systemd.unit.5)) is inspired by XDG Desktop Entry Specification .desktop files, which are in turn inspired by Microsoft Windows .ini files. Unit files are loaded from multiple locations (to see the full list, run `systemctl show --property=UnitPath`), but the main ones are (listed from lowest to highest precedence):

- `/usr/lib/systemd/system/` : units provided by installed packages
- `/etc/systemd/system/` : units installed by the system administrator

Note

- The load paths are completely different when running *systemd* in user mode.
- *systemd* unit names may only contain ASCII alphanumeric characters, underscores and periods. All other characters must be replaced by C-style "\x2d" escapes, or employ their predefined semantics ('@', '-'). See systemd.unit(5) (https://man.archlinux.org/man/systemd.unit.5) and systemd-escape(1) (https://man.archlinux.org/man/systemd-escape.1) for more information.

Look at the units installed by your packages for examples, as well as systemd.service(5) § EXAMPLES (https://man.archlinux.org/man/systemd.service.5#EXAMPLES).

Tip

Comments prepended with `#` may be used in unit-files as well, but only in new lines. Do not use end-line comments after *systemd* parameters or the unit will fail to activate.

systemd-analyze(1) (https://man.archlinux.org/man/systemd-analyze.1) can help verifying the work. See the `systemd-analyze verify FILE...` section of that page.

2.1 Handling dependencies

With *systemd*, dependencies can be resolved by designing the unit files correctly. The most typical case is when unit *A* requires unit *B* to be running before *A* is started. In that case add `Requires=B` and `After=B` to the `[Unit]` section of *A*. If the dependency is optional, add `Wants=B` and `After=B` instead. Note that `Wants=` and `Requires=` do not imply `After=`, meaning that if `After=` is not specified, the two units will be started in parallel.

Dependencies are typically placed on services and not on **#Targets**. For example, `network.target` is pulled in by whatever service configures your network interfaces, therefore ordering your custom unit after it is sufficient since `network.target` is started anyway.

2.2 Service types

There are several different start-up types to consider when writing a custom service file. This is set with the `Type=` parameter in the `[Service]` section:

- `Type=simple` (default): *systemd* considers the service to be started up immediately. The process must not fork. Do not use this type if other services need to be ordered on this service, unless it is socket activated.
- `Type=forking` : *systemd* considers the service started up once the process forks and the parent has exited. For classic daemons, use this type unless you know that it is not necessary. You should specify `PIDFile=` as well so *systemd* can keep track of the main process.
- `Type=oneshot` : this is useful for scripts that do a single job and then exit. You may want to set `RemainAfterExit=yes` as well so that *systemd* still considers the service as active after the process has exited. Setting `RemainAfterExit=yes` is appropriate for the units which change the system state (e.g., mount some partition). See also **[1] (https://trstringer.com/simple-vs-oneshot-systemd-service/)** for the differences of simple and oneshot.
- `Type=notify` : identical to `Type=simple` , but with the stipulation that the daemon will send a signal to *systemd* when it is ready. The reference implementation for this notification is provided by *libsystemd-daemon.so*.
- `Type=dbus` : the service is considered ready when the specified `BusName` appears on DBus's system bus.
- `Type=idle` : *systemd* will delay execution of the service binary until all jobs are dispatched unless these take longer than 5s, where the service binary is started anyway. Other than that its behaviour is very similar to `Type=simple` . It should never be used for service ordering and is intended for helping with console output readability.

See the **`systemd.service(5) § OPTIONS (https://man.archlinux.org/man/systemd.service.5#OPTIONS)`** man page for a more detailed explanation of the `Type` values.

2.3 Editing provided units

To avoid conflicts with pacman, unit files provided by packages should not be directly edited. There are two safe ways to modify a unit without touching the original file: create a new unit file which **overrides the original unit** or create **drop-in snippets** which are applied on top of the original unit. For both methods, you must reload the unit afterwards to apply your changes. This can be done either by editing the unit with `systemctl edit` (which reloads the unit automatically) or by reloading all units with:

```
# systemctl daemon-reload
```

Tip

- You can use *systemd-delta* to see which unit files have been overridden or extended and what exactly has been changed.

- Use `systemctl cat unit` to view the content of a unit file and all associated drop-in snippets.

2.3.1 Replacement unit files

To replace the unit file `/usr/lib/systemd/system/unit`, create the file `/etc/systemd/system/unit` and **reenable** the unit to update the symlinks.

Alternatively, run:

```
# systemctl edit --full unit
```

This opens `/etc/systemd/system/unit` in your editor (copying the installed version if it does not exist yet) and automatically reloads it when you finish editing.

> **Note**
>
> The replacement units will keep on being used even if Pacman updates the original units in the future. This method makes system maintenance more difficult and therefore the next approach is preferred.

2.3.2 Drop-in files

To create drop-in files for the unit file `/usr/lib/systemd/system/unit`, create the directory `/etc/systemd/system/unit.d/` and place *.conf* files there to override or add new options. *systemd* will parse and apply these files on top of the original unit.

The easiest way to do this is to run:

```
# systemctl edit unit --drop-in=drop_in_name
```

This opens the file `/etc/systemd/system/unit.d/drop_in_name.conf` in your text editor (creating it if necessary) and automatically reloads the unit when you are done editing. Omitting `--drop-in=` option will result in systemd using the default file name `override.conf`.

> **Note**
>
> - The key must be still placed in the appropriate section in the override file.
> - Not all keys can be overridden with drop-in files. For example, for changing `Conflicts=` a replacement file **is necessary (https://lists.freedesktop.org/archives/systemd-devel/2017-June/038976.html)**.
> - You can use top-level drop-in files to affect all units of the same type. For example, a drop-in file in `/etc/systemd/system/service.d/` affects all *.service* units. You can see an example in **#Notifying about failed services**

2.3.3 Revert to vendor version

To revert any changes to a unit made using `systemctl edit` do:

```
# systemctl revert unit
```

2.3.4 Examples

For example, if you simply want to add an additional dependency to a unit, you may create the following file:

```
/etc/systemd/system/unit.d/customdependency.conf
```

```
[Unit]
Requires=new dependency
After=new dependency
```

As another example, in order to replace the `ExecStart` directive, create the following file:

```
/etc/systemd/system/unit.d/customexec.conf
```

```
[Service]
ExecStart=
ExecStart=new command
```

Note how `ExecStart` must be cleared before being re-assigned **[2] (https://bugzilla.redhat.com/show_bug.cgi?id=756787#c9)**. The same holds for every item that can be specified multiple times, e.g. `OnCalendar` for timers.

One more example to automatically restart a service:

```
/etc/systemd/system/unit.d/restart.conf
```

```
[Service]
Restart=always
RestartSec=30
```

2.4 Service logging levels

For services that send logs directly to journald or syslog, you can control their verbosity by setting a numeric value between 0 and 6 for the `LogLevelMax=` parameter in the `[Service]` section using the methods described above. For example:

```
/etc/systemd/system/unit.d/override.conf
```

```
[Service]
LogLevelMax=3
```

The standard log levels are identical to those used to filter the **journal**. Setting a lower number excludes all higher and less important log messages from your journal.

2.4.1 Suppressing a service's standard output

If a service is echoing stdout and/or stderr output, by default this will end up in the journal as well. This behavior can be suppressed by setting `StandardOutput=null` and/or `StandardError=null` in the `[Service]` section. Other values than `null` can be used to further tweak this behavior. See [systemd.exec(5) § LOGGING_AND_STANDARD_INPUT/OUTPUT](https://man.archlinux.org/man/systemd.exec.5#LOGGING_AND_STANDARD_INPUT/OUTPUT).

3 Targets

systemd uses *targets* to group units together via dependencies and as standardized synchronization points. They serve a similar purpose as **runlevels** but act a little differently. Each *target* is named instead of numbered and is intended to serve a specific purpose with the possibility of having multiple ones active at the same time. Some *target*s are implemented by inheriting all of the services of another *target* and adding additional services to it. There are *systemd target*s that mimic the common SystemVinit runlevels.

3.1 Get current targets

The following should be used under *systemd* instead of running `runlevel`:

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

3.2 Create custom target

The runlevels that held a defined meaning under sysvinit (i.e., 0, 1, 3, 5, and 6); have a 1:1 mapping with a specific *systemd target*. Unfortunately, there is no good way to do the same for the user-defined runlevels like 2 and 4. If you make use of those it is suggested that you make a new named *systemd target* as `/etc/systemd/system/your target` that takes one of the existing runlevels as a base (you can look at `/usr/lib/systemd/system/graphical.target` as an example), make a directory `/etc/systemd/system/your target.wants`, and then symlink the additional services from `/usr/lib/systemd/system/` that you wish to enable.

3.3 Mapping between SysV runlevels and systemd targets

SysV Runlevel	systemd Target	Notes
0	poweroff.target	Halt the system.
1, s, single	rescue.target	Single user mode.
2, 4	multi-user.target	User-defined/Site-specific runlevels. By default, identical to 3.
3	multi-user.target	Multi-user, non-graphical. Users can usually login via multiple consoles or via the network.
5	graphical.target	Multi-user, graphical. Usually has all the services of runlevel 3 plus a graphical login.
6	reboot.target	Reboot
emergency	emergency.target	Emergency shell

3.4 Change current target

In *systemd*, targets are exposed via *target units*. You can change them like this:

```
# systemctl isolate graphical.target
```

This will only change the current target, and has no effect on the next boot. This is equivalent to commands such as `telinit 3` or `telinit 5` in Sysvinit.

3.5 Change default target to boot into

The standard target is `default.target`, which is a symlink to `graphical.target`. This roughly corresponds to the old runlevel 5.

To verify the current target with *systemctl*:

```
$ systemctl get-default
```

To change the default target to boot into, change the `default.target` symlink. With *systemctl*:

```
# systemctl set-default multi-user.target
```

```
Removed /etc/systemd/system/default.target.
Created symlink /etc/systemd/system/default.target -> /usr/lib/systemd/system/multi-user.target.
```

Alternatively, append one of the following **kernel parameters** to your boot loader:

- `systemd.unit=multi-user.target` (which roughly corresponds to the old runlevel 3),
- `systemd.unit=rescue.target` (which roughly corresponds to the old runlevel 1).

3.6 Default target order

systemd chooses the `default.target` according to the following order:

1. Kernel parameter shown above
2. Symlink of `/etc/systemd/system/default.target`
3. Symlink of `/usr/lib/systemd/system/default.target`

4 systemd components

Some (not exhaustive) components of *systemd* are:

- **kernel-install** — to automatically move **kernels** and their respective **initramfs** images to the boot partition;
- `systemd-analyze(1) (https://man.archlinux.org/man/systemd-analyze.1)` — may be used to determine boot-up performance, statistics and retrieve other state and tracing information, and to verify the correctness of unit files. It is also used to access special functions useful for advanced debugging.
- **systemd-boot** — simple UEFI **boot manager**;
- **systemd-creds** — to securely store and retrieve credentials used by systemd units;
- **systemd-cryptenroll** — Enroll PKCS#11, FIDO2, TPM2 token/devices to LUKS2 encrypted volumes;
- **systemd-firstboot** — basic system setting initialization before first boot;
- **systemd-homed** — portable human-user **accounts**;
- `systemd-logind(8) (https://man.archlinux.org/man/systemd-logind.8)` — **session management (https://dvdhrm.wordpress.com/2013/08/24/session-management-on-linux/)**;
- **systemd-networkd** — **network configuration** management;
- **systemd-nspawn** — light-weight namespace container;
- **systemd-repart** — creates partition tables, adds or grows partitions;
- **systemd-resolved** — network **name resolution**;
- `systemd-run(1) (https://man.archlinux.org/man/systemd-run.1)` / `run0(1) (https://man.archlinux.org/man/run0.1)` — Temporarily and interactively acquire elevated or different privileges.
- `systemd-stub(7) (https://man.archlinux.org/man/systemd-stub.7)` — a UEFI boot stub used for creating **unified kernel images**;
- `systemd-sysusers(8) (https://man.archlinux.org/man/systemd-sysusers.8)` — creates system users and groups and adds users to groups at package installation or boot time;
- **systemd-timesyncd** — **system time** synchronization across the network;
- **systemd/Journal** — system logging;
- **systemd/Timers** — monotonic or realtime timers for controlling *.service* files or events, reasonable alternative to **cron**.

4.1 systemd.mount - mounting

systemd is in charge of mounting the partitions and filesystems specified in `/etc/fstab`. The **systemd-fstab-generator(8) (https://man.archlinux.org/man/systemd-fstab-generator.8)** translates all the entries in `/etc/fstab` into *systemd* units; this is performed at boot time and whenever the configuration of the system manager is reloaded.

systemd extends the usual **fstab** capabilities and offers additional mount options. These affect the dependencies of the mount unit. They can, for example, ensure that a mount is performed only once the network is up or only once another partition is mounted. The full list of specific *systemd* mount options, typically prefixed with `x-systemd.`, is detailed in **systemd.mount(5) § FSTAB (https://man.archlinux.org/man/systemd.mount.5#FSTAB)**.

An example of these mount options is *automounting*, which means mounting only when the resource is required rather than automatically at boot time. This is provided in **fstab#Automount with systemd**.

4.1.1 GPT partition automounting

On UEFI-booted systems, GPT partitions such as `root`, `home`, `swap`, etc. can be mounted automatically following the **Discoverable Partitions Specification (https://uapi-group.org/specifications/specs/discoverable_partitions_specification/)**. These partitions can thus be omitted from **fstab**, and if the root partition is automounted, then `root=` can be omitted from the kernel command line. See **systemd-gpt-auto-generator(8) (https://man.archlinux.org/man/systemd-gpt-auto-generator.8)**.

The prerequisites are:

- When using **mkinitcpio**, the **systemd hook** is required.
- All automounted partitions must reside on the same physical disk as the ESP.
- The correct GPT partition types must be set. See **Partitioning#Partition scheme**.
- The boot loader must set the **LoaderDevicePartUUID (https://systemd.io/BOOT_LOADER_INTERFACE/)** EFI variable, so that the used EFI system partition can be identified. This is supported by **systemd-boot**, **systemd-stub(7) (https://man.archlinux.org/man/systemd-stub.7)**, **Limine**, **GRUB** (with *grub-mkconfig* generated `grub.cfg`; custom `grub.cfg` requires **loading the bli module**) and **rEFInd (not enabled by default)**. This can be verified by running `bootctl` and checking if there is a line with `Partition:` under `Current Boot Loader` or `Current Stub` when booting via **Unified kernel images**.

udev will create a `/dev/gpt-auto-root` symlink that points to the root volume block device. If the root partition is encrypted with LUKS, `/dev/gpt-auto-root` will point to the unlocked/mapped volume and `/dev/gpt-auto-root-luks` to the encrypted partition.

> **Tip**
>
> The automounting of a partition can be disabled by changing the partition's **type GUID** or setting the partition attribute bit 63 "do not automount", see **gdisk#Prevent GPT partition automounting**.

4.1.1.1 /var

For `/var` automounting to work, the PARTUUID must match the SHA256 HMAC hash of the partition type UUID keyed by the machine ID. The required PARTUUID can be obtained using:

```
$ systemd-id128 -u var-partition-uuid
```

Note

systemd-id128(1) (https://man.archlinux.org/man/systemd-id128.1) reads the machine ID from `/etc/machine-id`, this makes it impossible to know the needed PARTUUID before the system is installed.

4.2 systemd-sysvcompat

The primary role of systemd-sysvcompat (https://archlinux.org/packages/?name=systemd-sysvcompat) (required by base (https://archlinux.org/packages/?name=base)) is to provide the traditional linux init binary. For *systemd*-controlled systems, `init` is just a symbolic link to its `systemd` executable.

In addition, it provides four convenience shortcuts that SysVinit users might be used to. The convenience shortcuts are halt(8) (https://man.archlinux.org/man/halt.8), poweroff(8) (https://man.archlinux.org/man/poweroff.8), reboot(8) (https://man.archlinux.org/man/reboot.8) and shutdown(8) (https://man.archlinux.org/man/shutdown.8). Each one of those four commands is a symbolic link to `systemctl`, and is governed by *systemd* behavior. Therefore, the discussion at #Power management applies.

systemd-based systems can give up those System V compatibility methods by using the `init=` boot parameter (see, for example, /bin/init is in systemd-sysvcompat ? (https://bbs.archlinux.org/viewtopic.php?id=233387)) and *systemd* native `systemctl` command arguments.

4.3 systemd-tmpfiles - temporary files

systemd-tmpfiles creates, deletes and cleans up volatile and temporary files and directories. It reads configuration files in `/etc/tmpfiles.d/` and `/usr/lib/tmpfiles.d/` to discover which actions to perform. Configuration files in the former directory take precedence over those in the latter directory.

Configuration files are usually provided together with service files, and they are named in the style of `/usr/lib/tmpfiles.d/`*`program`*`.conf`. For example, the Samba daemon expects the directory `/run/samba` to exist and to have the correct permissions. Therefore, the samba (https://archlinux.org/packages/?name=samba) package ships with this configuration:

```
/usr/lib/tmpfiles.d/samba.conf
```

```
D /run/samba 0755 root root
```

Configuration files may also be used to write values into certain files on boot. For example, if you used `/etc/rc.local` to disable wakeup from USB devices with `echo USBE > /proc/acpi/wakeup`, you may use the following tmpfile instead:

```
/etc/tmpfiles.d/disable-usb-wake.conf

#    Path                  Mode UID  GID  Age Argument
w    /proc/acpi/wakeup     -    -    -    -   USBE
```

It is possible to write multiple lines to the same file, either with `\n` in the argument or using the `w+` type on multiple lines (including the first one) for *appending*:

```
/etc/tmpfiles.d/disable-usb-wake.conf

#    Path                  Mode UID  GID  Age Argument
w+   /proc/acpi/wakeup     -    -    -    -   USBE
w+   /proc/acpi/wakeup     -    -    -    -   LID0
```

See the **systemd-tmpfiles(8) (https://man.archlinux.org/man/systemd-tmpfiles.8)** and **tmpfiles.d(5) (https://man.archlinux.org/man/tmpfiles.d.5)** man pages for details.

> **Note**
>
> This method may not work to set options in `/sys` since the *systemd-tmpfiles-setup* service may run before the appropriate device modules are loaded. In this case, you could check whether the module has a parameter for the option you want to set with `modinfo module` and set this option with a **config file in /etc/modprobe.d**. Otherwise, you will have to write a **udev rule** to set the appropriate attribute as soon as the device appears.

4.4 Drop-in configuration files

Configuration files provided by packages should not be directly edited to avoid conflicts with pacman updates. For this, many (but not all) systemd packages provide a way to modify the configuration, but without touching the original file by creation of drop-in snippets. Check the package manual to see if drop-in configuration files are supported.

To create a drop-in configuration file for the unit file `/etc/systemd/unit.conf`, create the directory `/etc/systemd/unit.conf.d/` and place *.conf* files there to override or add new options. *systemd* will parse and apply these files on top of the original unit.

Check the overall configuration:

```
$ systemd-analyze cat-config systemd/unit.conf
```

The applied drop-in snippets file(s) and content will be listed at the end. **Restart** the service for the changes to take effect.

5 Tips and tricks

5.1 Socket activation

Some packages provide a *.socket* unit. For example, `cups` (https://archlinux.org/packages/?name=cups) provides a `cups.socket` unit[3] (https://0pointer.de/blog/projects/socket-activation2.html). If `cups.socket` is **enabled** (and `cups.service` is left disabled), *systemd* will not start CUPS immediately; it will just listen to the appropriate sockets. Then, whenever a program attempts to connect to one of these CUPS sockets, *systemd* will start `cups.service` and transparently hand over control of these ports to the CUPS process.

5.2 GUI configuration tools

- **systemadm** — Graphical browser for *systemd* units. It can show the list of units, possibly filtered by type.

  https://github.com/systemd/systemd-ui || `systemd-ui` (https://archlinux.org/packages/?name=systemd-ui)

- **systemdGenie** — *systemd* management utility based on KDE technologies.

  https://apps.kde.org/systemdgenie/ || `systemdgenie` (https://archlinux.org/packages/?name=systemdgenie)

5.3 Running services after the network is up

To delay a service until after the network is up, include the following dependencies in the *.service* file:

/etc/systemd/system/*foo*.service

```
[Unit]
...
Wants=network-online.target
After=network-online.target
...
```

The network wait service of the **network manager** in use must also be enabled so that `network-online.target` properly reflects the network status.

- If using **NetworkManager**, `NetworkManager-wait-online.service` should be enabled together with `NetworkManager.service`. Check if this is the case with `systemctl is-enabled NetworkManager-wait-online.service`. If it is not enabled, then **reenable** `NetworkManager.service`.
- In the case of **netctl**, **enable** the `netctl-wait-online.service` (unless you are using *netctl-auto*; see **FS#75836 (https://bugs.archlinux.org/task/75836)**).
- If using **systemd-networkd**, `systemd-networkd-wait-online.service` should be enabled together with `systemd-networkd.service`. Check if this is the case with

`systemctl is-enabled systemd-networkd-wait-online.service`. If it is not enabled, then **reenable** `systemd-networkd.service`.

For more detailed explanations, see the discussion in the **Network configuration synchronization points (https://systemd.io/NETWORK_ONLINE/#discussion)**.

If a service needs to perform DNS queries, it should additionally be ordered after `nss-lookup.target`:

/etc/systemd/system/*foo*.service

```
[Unit]
...
Wants=network-online.target
After=network-online.target nss-lookup.target
...
```

See **systemd.special(7) § Special Passive System Units (https://man.archlinux.org/man/systemd.special.7#Special_Passive_System_Units)**.

For `nss-lookup.target` to have any effect it needs a service that pulls it in via `Wants=nss-lookup.target` and orders itself before it with `Before=nss-lookup.target`. Typically this is done by local **DNS resolvers**.

Check which active service, if any, is pulling in `nss-lookup.target` with:

```
$ systemctl list-dependencies --reverse nss-lookup.target
```

5.4 Enable installed units by default

Arch Linux ships with `/usr/lib/systemd/system-preset/99-default.preset` containing `disable *`. This causes *systemctl preset* to disable all units by default, such that when a new package is installed, the user must manually enable the unit.

If this behavior is not desired, simply create a symlink from `/etc/systemd/system-preset/99-default.preset` to `/dev/null` in order to override the configuration file. This will cause *systemctl preset* to enable all units that get installed—regardless of unit type—unless specified in another file in one *systemctl preset*'s configuration directories. User units are not affected. See **systemd.preset(5) (https://man.archlinux.org/man/systemd.preset.5)** for more information.

> **Note**
>
> Enabling all units by default may cause problems with packages that contain two or more mutually exclusive units. *systemctl preset* is designed to be used by distributions and spins or system administrators. In the case where two conflicting units would be enabled, you should explicitly specify which one is to be disabled in a preset configuration file as specified in the **systemd.preset(5) (https://man.archlinux.org/man/systemd.preset.5)** man page.

5.5 Sandboxing application environments

See **systemd/Sandboxing**.

5.6 Notifying about failed services

In order to notify about service failures, an `OnFailure=` directive needs to be added to the according service file, for example by using a **drop-in configuration file**. Adding this directive to every service unit can be achieved with a top-level drop-in configuration file. For details about top-level drop-ins, see **systemd.unit(5) (https://man.archlinux.org/man/systemd.unit.5)**.

Create a top-level drop-in for services:

```
/etc/systemd/system/service.d/toplevel-override.conf
---
[Unit]
OnFailure=failure-notification@%n.service
```

This adds `OnFailure=failure-notification@%n.service` to every service file. If *some_service_unit* fails, `failure-notification@some_service_unit.service` will be started to handle the notification delivery (or whatever task it is configured to perform).

Create the `failure-notification@.service` template unit:

```
/etc/systemd/system/failure-notification@.service
---
[Unit]
Description=Send a notification about a failed systemd unit

[Service]
Type=oneshot
ExecStart=/path/to/failure-notification.sh %i
# runs as a temporary user/group and enables several other security precautions
DynamicUser=true
```

You can create the `failure-notification.sh` script and define what to do or how to notify. Examples include **sending e-mail**, **showing desktop notifications**, using gotify, XMPP, etc. The `%i` will be the name of the failed service unit and will be passed as an argument to the script.

In order to prevent a recursion for starting instances of `failure-notification@.service` again and again if the start fails, create an empty drop-in configuration file with the same name as the top-level drop-in (the empty service-level drop-in configuration file takes precedence over the top-level drop-in and overrides the latter one):

```
# mkdir /etc/systemd/system/failure-notification@.service.d
# touch /etc/systemd/system/failure-notification@.service.d/toplevel-override.conf
```

5.7 Notifying with e-mail

You can set up systemd to send an e-mail when a unit fails. Cron sends mail to `MAILTO` if the job outputs to stdout or stderr, but many jobs are setup to only output on error. First you need two files: an executable for sending the mail and a *.service* for starting the executable. For this example, the executable is just a shell script using `sendmail`, which is in packages that provide `smtp-forwarder`.

/usr/local/bin/systemd-email

```
#!/bin/sh

/usr/bin/sendmail -t <<ERRMAIL
To: $1
From: systemd <root@$HOSTNAME>
Subject: $2
Content-Transfer-Encoding: 8bit
Content-Type: text/plain; charset=UTF-8

$(systemctl status --full "$2")
ERRMAIL
```

Whatever executable you use, it should probably take at least two arguments as this shell script does: the address to send to and the unit file to get the status of. The *.service* we create will pass these arguments:

/etc/systemd/system/status_email_*user*@.service

```
[Unit]
Description=status email for %i to user

[Service]
Type=oneshot
ExecStart=/usr/local/bin/systemd-email address %i
User=nobody
Group=systemd-journal
```

Where *`user`* is the user being emailed and *`address`* is that user's email address. Although the recipient is hard-coded, the unit file to report on is passed as an instance parameter, so this one service can send email for many other units. At this point you can **start** `status_email_user@dbus.service` to verify that you can receive the emails.

Then simply **edit** the service you want emails for and add `OnFailure=status_email_user@%n.service` to the `[Unit]` section. `%n` passes the unit's name to the template.

Note

- If you set up sSMTP security according to **sSMTP#Security** the user `nobody` will not have access to `/etc/ssmtp/ssmtp.conf`, and the `systemctl start status_email_user@dbus.service` command will fail. One solution is to use `root` as the User in the `status_email_user@.service` unit.
- If you try to use `mail -s somelogs address` in your email script, `mail` will fork and systemd will kill the mail process when it sees your script exit. Make the mail non-forking by doing `mail -Ssendwait -s somelogs address`.

Tip

Newer versions of systemd recommend using `DynamicUser=true` as a replacement for `User=nobody` which is now discouraged. See **GitHub issue 428 (https://github.com/v2fly/v2ray-core/issues/428)** for more details.

5.8 Automatically turn off an external HDD at shutdown

See **udisks#Automatically turn off an external HDD at shutdown**.

6 Troubleshooting

6.1 Investigating failed services

To find the *systemd* services which failed to start:

```
$ systemctl --state=failed
```

To find out why they failed, examine their log output. See **systemd/Journal#Filtering output** for details.

6.2 Diagnosing boot problems

systemd has several options for diagnosing problems with the boot process. See **boot debugging** for more general instructions and options to capture boot messages before *systemd* takes over the **boot process**. Also see **systemd debugging documentation (https://systemd.io/DEBUGGING/)**.

6.3 Diagnosing a service

If some *systemd* service misbehaves or you want to get more information about what is happening, set the `SYSTEMD_LOG_LEVEL` **environment variable** to `debug`. For example, to run the *systemd-networkd* daemon in debug mode:

Add a **drop-in file** for the service adding the two lines:

```
[Service]
Environment=SYSTEMD_LOG_LEVEL=debug
```

Or equivalently, set the environment variable manually:

```
# SYSTEMD_LOG_LEVEL=debug /lib/systemd/systemd-networkd
```

then **restart** *systemd-networkd* and watch the journal for the service with the `-f` / `--follow` option.

6.4 Shutdown/reboot takes terribly long

If the shutdown process takes a very long time (or seems to freeze), most likely a service not exiting is to blame. *systemd* waits some time for each service to exit before trying to kill it. To find out whether you are affected, see **Shutdown completes eventually (https://systemd.io/DEBUGGING/#shutdown-completes-eventually)** in the *systemd* documentation.

A common problem is a stalled shutdown or suspend process. To verify whether that is the case, you could run either of these commands and check the outputs

```
# systemctl poweroff
```

```
Failed to power off system via logind: There's already a shutdown or sleep operation in progress
```

```
# systemctl list-jobs
```

```
  JOB UNIT                    TYPE  STATE
...
21593 systemd-suspend.service start running
21592 suspend.target          start waiting
..
```

The **solution (https://unix.stackexchange.com/a/579531)** to this would be to cancel these jobs by running

```
# systemctl cancel
# systemctl stop systemd-suspend.service
```

and then trying shutdown or reboot again.

6.5 Short lived processes do not seem to log any output

If running `journalctl -u foounit` as root does not show any output for a short lived service, look at the PID instead. For example, if `systemd-modules-load.service` fails, and `systemctl status systemd-modules-load` shows that it ran as PID 123, then you might be able to see output in the journal for that PID, i.e. by running `journalctl -b _PID=123` as root. Metadata fields for the journal such as `_SYSTEMD_UNIT` and `_COMM` are collected asynchronously and rely on the `/proc` directory for the process existing. Fixing this requires fixing the kernel to provide this data via a socket connection, similar to `SCM_CREDENTIALS`. In short, it is a **bug (https://github.com/systemd/systemd/issues/2913)**. Keep in mind that immediately failed services might not print anything to the journal as per design of *systemd*.

6.6 Boot time increasing over time

After using `systemd-analyze` a number of users have noticed that their boot time has increased significantly in comparison with what it used to be. After using `systemd-analyze blame` **NetworkManager** is being reported as taking an unusually large amount of time to start.

The problem for some users has been due to `/var/log/journal` becoming too large. This may have other impacts on performance, such as for `systemctl status` or `journalctl`. As such the solution is to remove every file within the folder (ideally making a backup of it somewhere, at least temporarily) and then setting a journal file size limit as described in **systemd/Journal#Journal size limit**.

6.7 systemd-tmpfiles-setup.service fails to start at boot

Starting with *systemd* 219, `/usr/lib/tmpfiles.d/systemd.conf` specifies ACL attributes for directories under `/var/log/journal` and, therefore, requires ACL support to be enabled for the filesystem the journal resides on.

See **Access Control Lists#Enable ACL** for instructions on how to enable ACL on the filesystem that houses `/var/log/journal`.

6.8 Disable emergency mode on remote machine

You may want to disable emergency mode on a remote machine, for example, a virtual machine hosted at Azure or Google Cloud. It is because if emergency mode is triggered, the machine will be blocked from connecting to network.

To disable it, **mask** `emergency.service` and `emergency.target`.

6.9 Error "Unit xxx.service not found", but service does exist

You may be trying to start or enable a user unit as a system unit. **systemd.unit(5) (https://man.archlinux.org/man/systemd.unit.5)** indicates, which units reside where. By default *systemctl* operates on system services.

See **systemd/User** for more details.

6.10 Manually renewing LoaderDevicePartUUID after changing EFI partition UUID

Some bootloaders only set the `LoaderDevicePartUUID` variable when it is empty. Consequently, even if the UUID of the EFI partition changes, the bootloader will not update `LoaderDevicePartUUID`. By deleting the EFI variable with the commands below, the bootloader will then repopulate it with the new UUID.

```
# chattr -i /sys/firmware/efi/efivars/LoaderDevicePartUUID-4a67b082-0a4c-41cf-b6c7-440b29bb8c4f
# rm /sys/firmware/efi/efivars/LoaderDevicePartUUID-4a67b082-0a4c-41cf-b6c7-440b29bb8c4f
```

7 See also

- **Wikipedia:systemd**
- **Official web site (https://systemd.io/)**
  - **systemd optimizations (https://systemd.io/OPTIMIZATIONS/)**

- systemd FAQ (https://systemd.io/FAQ/)
- systemd Tips and tricks (https://systemd.io/TIPS_AND_TRICKS/)

- systemd(1) (https://man.archlinux.org/man/systemd.1)
- Other distributions
  - Gentoo:systemd
  - Fedora:systemd
  - Fedora:How to debug Systemd problems
  - Fedora:SysVinit to Systemd Cheatsheet
  - Debian:systemd
- Lennart's blog story (http://0pointer.de/blog/projects/systemd.html), update 1 (http://0pointer.de/blog/projects/systemd-update.html), update 2 (http://0pointer.de/blog/projects/systemd-update-2.html), update 3 (http://0pointer.de/blog/projects/systemd-update-3.html), summary (http://0pointer.de/blog/projects/why.html)
- Debug Systemd Services (https://containersolutions.github.io/runbooks/posts/linux/debug-systemd-service-units)
- systemd for Administrators (PDF) (http://0pointer.de/public/systemd-ebook-psankar.pdf)
- How To Use Systemctl to Manage Systemd Services and Units (https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units)
- Session management with systemd-logind (https://dvdhrm.wordpress.com/2013/08/24/session-management-on-linux/)
- Emacs#Syntax highlighting for systemd files
- Two (https://www.h-online.com/open/features/Control-Centre-The-systemd-Linux-init-system-1565543.html) part (https://www.h-online.com/open/features/Booting-up-Tools-and-tips-for-systemd-1570630.html) introductory article in *The H Open* magazine.

Retrieved from "https://wiki.archlinux.org/index.php?title=Systemd&oldid=846515"