systemd

From the **project web page (http://freedesktop.org/wiki/Software/systemd)**:

> *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 FAQ
- init
- Daemons#List of daemons
- udev
- Improve boot performance
- Allow users to shutdown

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

Contents

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://jlk.fjfi.cvut.cz/arch/manpages/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.
- *systemadm* is the official graphical frontend for *systemctl* and is provided by the `systemd-ui (https://www.archlinux.org/packages/?name=systemd-ui)` package.
- **Plasma** users can install `systemd-kcm (https://www.archlinux.org/packages/?name=systemd-kcm)` as a graphical frontend for *systemctl*. After installing the module will be added under *System administration*.

Analyzing the system state

Show **system status** using:

```
$ systemctl status
```

List running units:

```
$ systemctl
```

or:

```
$ systemctl list-units
```

List failed units:

```
$ systemctl --failed
```

The available unit files can be seen in `/usr/lib/systemd/system/` and `/etc/systemd/system/` (the latter takes precedence). **List installed** unit files with:

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

Using units

Units can be, for example, services (*.service*), mount points (*.mount*), devices (*.device*) or 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://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.unit.5)** for details.

Note: Some unit names contain an `@` sign (e.g. `name@*string*.service`): this means that they are **instances (http://0pointer.de/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 just fail, since the `%i` specifier cannot be substituted.

Tip:

- Most of the following commands also work if multiple units are specified, see systemctl(1) (https://jlk.fjfi.cvut.cz/arch/manpages/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 immediately the unit rather than after the next boot.
- 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.

Start a unit immediately:

```
# systemctl start unit
```

Stop a unit immediately:

```
# systemctl stop unit
```

Restart a unit:

```
# systemctl restart unit
```

Ask a unit to **reload** its configuration:

```
# systemctl reload unit
```

Show the **status** of a unit, including whether it is running or not:

```
$ systemctl status unit
```

Check whether a unit is already enabled or not:

```
$ systemctl is-enabled unit
```

Enable a unit to be started on **bootup**:

```
# systemctl enable unit
```

Enable a unit to be started on **bootup** and **Start** immediately:

```
# systemctl enable --now unit
```

Disable a unit to not start during bootup:

```
# systemctl disable unit
```

Mask a unit to make it impossible to start it:

```
# systemctl mask unit
```

Unmask a unit:

```
# systemctl unmask unit
```

Show the **manual page** associated with a unit (this has to be supported by the unit file):

```
$ systemctl help unit
```

Reload *systemd*, scanning for **new or changed units**:

```
# systemctl daemon-reload
```

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.

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:

```
$ systemctl hibernate
```

Put the system into hybrid-sleep state (or suspend-to-both):

```
$ systemctl hybrid-sleep
```

Writing unit files

The syntax of *systemd*'s **unit files (http://www.freedesktop.org/software/systemd/man/systemd.unit.html)** is inspired by XDG Desktop Entry Specification *.desktop* files, which are in turn inspired by Microsoft Windows *.ini* files. Unit files are loaded from two locations. From lowest to highest precedence they are:

- `/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 pre defined semantics ('@', '-'). See `systemd.unit(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.unit.5)` and `systemd-escape(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd-escape.1)` for more information.

Look at the units installed by your packages for examples, as well as the **annotated example section (http://www.freedesktop.org/software/systemd/man/systemd.service.html#Examples)** of `systemd.service(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.service.5)`.

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.

Handling dependencies

With *systemd*, dependencies can be resolved by designing the unit files correctly. The most typical case is that the unit *A* requires the 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.

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.
- `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. Other than that behavior is very similar to `Type=simple`.

See the **systemd.service(5) (http://www.freedesktop.org/software/systemd/man/systemd.service.html#Type=)** man page for a more detailed explanation of the `Type` values.

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.
- The default syntax highlighting for *systemd* unit files within **Vim** is the same as for **INI files**. However, if you want something more systemd-specific, install `vim-systemd (https://www.archlinux.org/packages/?name=vim-systemd)`.

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:

```
# systemctl reenable unit
```

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.

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
```

This opens the file `/etc/systemd/system/unit.d/override.conf` in your text editor (creating it if necessary) and automatically reloads the unit when you are done editing.

Revert to vendor version

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

```
# systemctl revert unit
```

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 for a unit that is not of type `oneshot`, 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 **[1] (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
```

Targets

systemd uses *targets* which serve a similar purpose as **runlevels** but act a little different. 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 so you can still switch *target*s using the familiar `telinit RUNLEVEL` command.

Get current targets

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

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

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.

Mapping between SysV runlevels and systemd targets

SysV Runlevel	systemd Target	Notes
0	runlevel0.target, poweroff.target	Halt the system.
1, s, single	runlevel1.target, rescue.target	Single user mode.
2, 4	runlevel2.target, runlevel4.target, multi-user.target	User-defined/Site-specific runlevels. By default, identical to 3.
3	runlevel3.target, multi-user.target	Multi-user, non-graphical. Users can usually login via multiple consoles or via the network.
5	runlevel5.target, graphical.target	Multi-user, graphical. Usually has all the services of runlevel 3 plus a graphical login.
6	runlevel6.target, reboot.target	Reboot
emergency	emergency.target	Emergency shell

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.

Change default target to boot into

The standard target is `default.target`, which is aliased by default to `graphical.target` (which roughly corresponds to the old runlevel 5). To change the default target at boot-time, append one of the following **kernel parameters** to your bootloader:

- `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).

Alternatively, you may leave the bootloader alone and change `default.target`. This can be done using *systemctl*:

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

To be able to override the previously set `default.target`, use the force option:

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

The effect of this command is output by *systemctl*; a symlink to the new default target is made at `/etc/systemd/system/default.target`.

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://www.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
```

```
w /proc/acpi/wakeup - - - - USBE
```

See the systemd-tmpfiles(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd-tmpfiles.8) and tmpfiles.d(5) (https://jlk.fjfi.cvut.cz/arch/manpages/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 is 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.

Timers

A timer is a unit configuration file whose name ends with *.timer* and encodes information about a timer controlled and supervised by *systemd*, for timer-based activation. See **systemd/Timers**.

Note: Timers can replace **cron** functionality to a great extent. See **systemd/Timers#As a cron replacement**.

Mounting

systemd is in charge of mounting the partitions and filesystems specified in `/etc/fstab`. The **`systemd-fstab-generator(8) (https://jlk.fjfi.cvut.cz/arch/manpages/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) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.mount.5#FSTAB)`**.

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

Journal

systemd has its own logging system called the journal; therefore, running a `syslog` daemon is no longer required. To read the log, use:

```
# journalctl
```

In Arch Linux, the directory `/var/log/journal/` is a part of the systemd (https://www.archlinux.org/packages/?name=systemd) package, and the journal (when `Storage=` is set to `auto` in `/etc/systemd/journald.conf`) will write to `/var/log/journal/`. If you or some program delete that directory, *systemd* will **not** recreate it automatically and instead will write its logs to `/run/systemd/journal` in a nonpersistent way. However, the folder will be recreated when you set `Storage=persistent` and run `systemctl restart systemd-journald` (or reboot).

Systemd journal classifies messages by **Priority level** and **Facility**. Logging classification corresponds to classic **Syslog** protocol (**RFC 5424 (https://tools.ietf.org/html/rfc5424)**).

Priority level

A syslog severity code (in systemd called priority) is used to mark the importance of a message **RFC 5424 Section 6.2.1 (https://tools.ietf.org/html/rfc5424#section-6.2.1)**.

Value	Severity	Keyword	Description	Examples
0	Emergency	emerg	System is unusable	Severe Kernel BUG, systemd dumped core. This level should not be used by applications.
1	Alert	alert	Should be corrected immediately	Vital subsystem goes out of work. Data loss. `kernel: BUG: unable to handle kernel paging request at ffffc90403238ffc`.
2	Critical	crit	Critical conditions	Crashes, coredumps. Like familiar flash: `systemd-coredump[25319]: Process 25310 (plugin-containe) of user 1000 dumped core` Failure in the system primary application, like X11.
3	Error	err	Error conditions	Not severe error reported: `kernel: usb 1-3: 3:1: cannot get freq at ep 0x84`, `systemd[1]: Failed unmounting /var.`, `libvirtd[1720]: internal error: Failed to initialize a valid firewall backend`).
4	Warning	warning	May indicate that an error will occur if action is not taken.	A non-root file system has only 1GB free. `org.freedesktop. Notifications[1860]: (process:5999): Gtk-WARNING **: Locale not supported by C library. Using the fallback 'C' locale` .
5	Notice	notice	Events that are unusual, but not error conditions.	`systemd[1]: var.mount: Directory /var to mount over is not empty, mounting anyway`. `gcr-prompter[4997]: Gtk: GtkDialog mapped without a transient parent. This is discouraged` .
6	Informational	info	Normal operational messages that require no action.	`lvm[585]: 7 logical volume(s) in volume group "archvg" now active`.
7	Debug	debug	Information useful to developers for debugging the application.	`kdeinit5[1900]: powerdevil: Scheduling inhibition from ":1.14" "firefox" with cookie 13 and reason "screen"` .

If you cannot find a message on the expected priority level, also search a couple of levels above and below: these rules are recommendations, and the developer of the affected application may have a different perception of the issue's importance from yours.

Facility

A syslog facility code is used to specify the type of program that is logging the message **RFC 5424 Section 6.2.1 (https://tools.ietf.org/html/rfc5424#section-6.2.1)**.

Facility code	Keyword	Description	Info
0	kern	kernel messages	
1	user	user-level messages	
2	mail	mail system	Archaic POSIX still supported and sometimes used (for more `mail(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/mail.1)`)
3	daemon	system daemons	All daemons, including systemd and its subsystems
4	auth	security/authorization messages	Also watch for different facility 10
5	syslog	messages generated internally by syslogd	As it standartized for syslogd, not used by systemd (see facility 3)
6	lpr	line printer subsystem (archaic subsystem)	
7	news	network news subsystem (archaic subsystem)	
8	uucp	UUCP subsystem (archaic subsystem)	
9		clock daemon	systemd-timesyncd
10	authpriv	security/authorization messages	Also watch for different facility 4
11	ftp	FTP daemon	
12	-	NTP subsystem	
13	-	log audit	
14	-	log alert	
15	cron	scheduling daemon	
16	local0	local use 0 (local0)	
17	local1	local use 1 (local1)	
18	local2	local use 2 (local2)	
19	local3	local use 3 (local3)	
20	local4	local use 4 (local4)	
21	local5	local use 5 (local5)	
22	local6	local use 6 (local6)	
23	local7	local use 7 (local7)	

So, useful facilities to watch: 0,1,3,4,9,10,15.

Filtering output

journalctl allows you to filter the output by specific fields. Be aware that if there are many messages to display or filtering of large time span has to be done, the output of this command can be delayed for quite some time.

Tip: While the journal is stored in a binary format, the content of stored messages is not modified. This means it is viewable with *strings*, for example for recovery in an environment which does not have *systemd* installed. Example command:

```
$ strings /mnt/arch/var/log/journal/af4967d77fba44c6b093d0e9862f6ddd/system.journal | grep -i message
```

Examples:

- Show all messages from this boot:

  ```
  # journalctl -b
  ```

  However, often one is interested in messages not from the current, but from the previous boot (e.g. if an unrecoverable system crash happened). This is possible through optional offset parameter of the `-b` flag: `journalctl -b -0` shows messages from the current boot, `journalctl -b -1` from the previous boot, `journalctl -b -2` from the second previous and so on – you can see the list of boots with their numbers by using `journalctl --list-boots`. See journalctl(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/journalctl.1) for full description, the semantics is much more powerful.

- Show all messages from date (and optional time):

```
# journalctl --since="2012-10-30 18:17:16"
```

- Show all messages since 20 minutes ago:

```
# journalctl --since "20 min ago"
```

- Follow new messages:

```
# journalctl -f
```

- Show all messages by a specific executable:

```
# journalctl /usr/lib/systemd/systemd
```

- Show all messages by a specific process:

```
# journalctl _PID=1
```

- Show all messages by a specific unit:

```
# journalctl -u netcfg
```

- Show kernel ring buffer:

```
# journalctl -k
```

- Show only error, critical, and alert priority messages

  ```
  # journalctl -p err..alert
  ```

  Numbers also can be used, `journalctl -p 3..1`. If single number/keyword used, `journalctl -p 3` - all higher priority levels also included.
- Show auth.log equivalent by filtering on syslog facility:

  ```
  # journalctl SYSLOG_FACILITY=10
  ```

See **journalctl(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/journalctl.1)**, **systemd.journal-fields(7) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd.journal-fields.7)**, or Lennart's **blog post (http://0pointer.de/blog/projects/journalctl.html)** for details.

Tip: By default, *journalctl* truncates lines longer than screen width, but in some cases, it may be better to enable wrapping instead of truncating. This can be controlled by the `SYSTEMD_LESS` **environment variable**, which contains options passed to **less** (the default pager) and defaults to `FRSXMK` (see **less(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/less.1)** and **journalctl(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/journalctl.1)** for details).

By omitting the `S` option, the output will be wrapped instead of truncated. For example, start *journalctl* as follows:

```
$ SYSTEMD_LESS=FRXMK journalctl
```

If you would like to set this behaviour as default, **export** the variable from `~/.bashrc` or `~/.zshrc`.

Journal size limit

If the journal is persistent (non-volatile), its size limit is set to a default value of 10% of the size of the underlying file system but capped to 4 GiB. For example, with `/var/log/journal/` located on a 20 GiB partition, journal data may take up to 2 GiB. On a 50 GiB partition, it would max at 4 GiB.

The maximum size of the persistent journal can be controlled by uncommenting and changing the following:

```
/etc/systemd/journald.conf
```

```
SystemMaxUse=50M
```

It is also possible to use the drop-in snippets configuration override mechanism rather than editing the global configuration file. In this case do not forget to place the overrides under the `[Journal]` header:

```
/etc/systemd/journald.conf.d/00-journal-size.conf
```

```
[Journal]
SystemMaxUse=50M
```

Restart the `systemd-journald.service` after changing this setting to immediately apply the new limit.

See **journald.conf(5) (https://jlk.fjfi.cvut.cz/arch/manpages/man/journald.conf.5)** for more info.

Clean journal files manually

Journal files can be globally removed from `/var/log/journal/` using *e.g.* `rm`, or can be trimmed according to various criteria using `journalctl`. Examples:

- Remove archived journal files until the disk space they use falls below 100M:

```
# journalctl --vacuum-size=100M
```

- Make all journal files contain no data older than 2 weeks.

```
# journalctl --vacuum-time=2weeks
```

See **journalctl(1) (https://jlk.fjfi.cvut.cz/arch/manpages/man/journalctl.1)** for more info.

Journald in conjunction with syslog

Compatibility with a classic, non-journald aware syslog implementation can be provided by letting *systemd* forward all messages via the socket `/run/systemd/journal/syslog`. To make the syslog daemon work with the journal, it has to bind to this socket instead of `/dev/log` (official announcement (http://lwn.net/Articles/474968/)).

The default `journald.conf` for forwarding to the socket is `ForwardToSyslog=no` to avoid system overhead, because rsyslog or syslog-ng pull the messages from the journal by itself (http://lists.freedesktop.org/archives/systemd-devel/2014-August/022295.html#journald).

See Syslog-ng#Overview and Syslog-ng#syslog-ng and systemd journal, or rsyslog respectively, for details on configuration.

Forward journald to /dev/tty12

Create a drop-in directory `/etc/systemd/journald.conf.d` and create a `fw-tty12.conf` file in it:

```
/etc/systemd/journald.conf.d/fw-tty12.conf
```

```
[Journal]
ForwardToConsole=yes
TTYPath=/dev/tty12
MaxLevelConsole=info
```

Then restart systemd-journald.

Specify a different journal to view

There may be a need to check the logs of another system that is dead in the water, like booting from a live system to recover a production system. In such case, one can mount the disk in e.g. `/mnt`, and specify the journal path via `-D`/`--directory`, like so:

```
$ journalctl -D /mnt/var/log/journal -xe
```

Tips and tricks

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://jlk.fjfi.cvut.cz/arch/manpages/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 manpage for `systemd.preset`.

Sandboxing application environments

A unit file can be created as a sandbox to isolate applications and their processes within a hardened virtual environment. systemd leverages **namespaces**, white-/blacklisting of **Capabilities**, and **control groups** to container processes through an extensive **execution environment configuration (https://www.freedesktop.org/software/systemd/man/systemd.exec.html)**.

The enhancement of an existing systemd unit file with application sandboxing typically requires trial-and-error tests accompanied by the generous use of **`strace` (https://www.archlinux.org/packages/?name=strace)**, **stderr** and **journalctl (https://www.freedesktop.org/software/systemd/man/journalctl.html)** error logging and output facilities. You may want to first search upstream documentation for already done tests to base trials on.

Some examples on how sandboxing with systemd can be deployed:

- `CapabilityBoundingSet` defines a whitelisted set of allowed capabilities, but may also be used to blacklist a specific capability for a unit.

- The `CAP_SYS_ADM` capability, for example, which should be one of the **goals of a secure sandbox (https://lwn.net/Articles/486306/)**: `CapabilityBoundingSet=~ CAP_SYS_ADM`

Troubleshooting

Investigating systemd errors

As an example, we will investigate an error with `systemd-modules-load` service:

1. Lets find the *systemd* services which fail to start at boot time:

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

```
systemd-modules-load.service   loaded failed failed  Load Kernel Modules
```

Another way is to live log *systemd* messages:

```
$ journalctl -fp err
```

2. Ok, we found a problem with `systemd-modules-load` service. We want to know more:

```
$ systemctl status systemd-modules-load
```

```
systemd-modules-load.service - Load Kernel Modules
   Loaded: loaded (/usr/lib/systemd/system/systemd-modules-load.service; static)
   Active: failed (Result: exit-code) since So 2013-08-25 11:48:13 CEST; 32s ago
     Docs: man:systemd-modules-load.service(8).
           man:modules-load.d(5)
  Process: 15630 ExecStart=/usr/lib/systemd/systemd-modules-load (code=exited, status=1/FAILURE)
```

If the `Process ID` is not listed, just restart the failed service with `systemctl restart systemd-modules-load`

3. Now we have the process id (PID) to investigate this error in depth. Enter the following command with the current `Process ID` (here: 15630):

```
$ journalctl _PID=15630
```

```
-- Logs begin at Sa 2013-05-25 10:31:12 CEST, end at So 2013-08-25 11:51:17 CEST. --
Aug 25 11:48:13 mypc systemd-modules-load[15630]: Failed to find module 'blacklist usblp'
Aug 25 11:48:13 mypc systemd-modules-load[15630]: Failed to find module 'install usblp /bin/false'
```

4. We see that some of the kernel module configs have wrong settings. Therefore we have a look at these settings in `/etc/modules-load.d/`:

```
$ ls -Al /etc/modules-load.d/
```

```
...
-rw-r--r--   1 root root    79  1. Dez 2012  blacklist.conf
-rw-r--r--   1 root root     1  2. Mär 14:30 encrypt.conf
-rw-r--r--   1 root root     3  5. Dez 2012  printing.conf
-rw-r--r--   1 root root     6 14. Jul 11:01 realtek.conf
-rw-r--r--   1 root root    65  2. Jun 23:01 virtualbox.conf
...
```

5. The `Failed to find module 'blacklist usblp'` error message might be related to a wrong setting inside of `blacklist.conf`. Lets deactivate it with inserting a trailing # before each option we found via step 3:

```
/etc/modules-load.d/blacklist.conf
---
# blacklist usblp
# install usblp /bin/false
```

6. Now, try to start `systemd-modules-load`:

```
$ systemctl start systemd-modules-load
```

If it was successful, this should not prompt anything. If you see any error, go back to step 3 and use the new PID for solving the errors left.

If everything is ok, you can verify that the service was started successfully with:

```
$ systemctl status systemd-modules-load
---
systemd-modules-load.service - Load Kernel Modules
   Loaded: loaded (/usr/lib/systemd/system/systemd-modules-load.service; static)
   Active: active (exited) since So 2013-08-25 12:22:31 CEST; 34s ago
     Docs: man:systemd-modules-load.service(8)
           man:modules-load.d(5)
 Process: 19005 ExecStart=/usr/lib/systemd/systemd-modules-load (code=exited, status=0/SUCCESS)
Aug 25 12:22:31 mypc systemd[1]: Started Load Kernel Modules.
```

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 the **systemd debugging documentation (http://freedesktop.org/wiki/Software/systemd/Debugging/)**.

Diagnosing problems with a specific service

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

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

Or, equivalently, modify the service file temporarily for gathering enough output. For example:

```
/usr/lib/systemd/system/systemd-networkd.service
```

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

If debug information is required long-term, add the variable the **regular** way.

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 if you are affected, see **this article (http://freedesktop.org/wiki/Software/systemd/Debugging/#shutdowncompleteseventually)**.

Short lived processes do not seem to log any output

If `journalctl -u foounit` 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. `journalctl -b _PID=123`. 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`.

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 **#Journal size limit**.

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#Enabling ACL** for instructions on how to enable ACL on the filesystem that houses `/var/log/journal`.

systemd version printed on boot is not the same as installed package version

You need to **regenerate your initramfs** and the versions should match.

Tip: A pacman hook can be used to automatically regenerate the initramfs every time **`systemd (https://www.archlinux.org/packages/?name=systemd)`** is upgraded. See **this forum thread (https://bbs.archlinux.org/viewtopic.php?id=215411)** and **Pacman#Hooks**.

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.

```
# systemctl mask emergency.service
# systemctl mask emergency.target
```

See also

- Wikipedia article
- systemd Official web site (http://www.freedesktop.org/wiki/Software/systemd)
- systemd optimizations (http://www.freedesktop.org/wiki/Software/systemd/Optimizations)
- systemd FAQ (http://www.freedesktop.org/wiki/Software/systemd/FrequentlyAskedQuestions)
- systemd Tips and tricks (http://www.freedesktop.org/wiki/Software/systemd/TipsAndTricks)
- Gentoo Wiki systemd page (http://wiki.gentoo.org/wiki/Systemd)
- Fedora Project - About systemd (http://fedoraproject.org/wiki/Systemd)
- Fedora Project - How to debug systemd problems (http://fedoraproject.org/wiki/How_to_debug_Systemd_problems)

- Fedora Project - SysVinit to systemd cheatsheet (http://fedoraproject.org/wiki/SysVinit_to_Systemd_Cheatsheet)
- Gentoo Wiki systemd page
- Debian Wiki systemd page
- Manual pages (http://0pointer.de/public/systemd-man/)
- 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)
- 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 (http://www.h-online.com/open/features/Control-Centre-The-systemd-Linux-init-system-1565543.html) part (http://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=510462#Using_units"