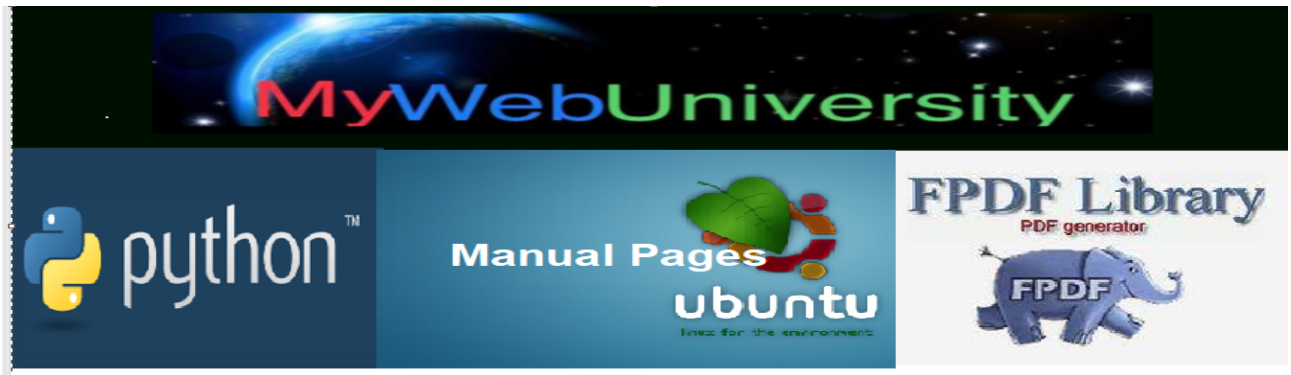

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'systemd-notify.1'

$ man systemd-notify.1

SYSTEMD-NOTIFY(1) systemd-notify SYSTEMD-NOTIFY(1)

NAME

systemd-notify - Notify service manager about start-up completion and other daemon status changes

SYNOPSIS

systemd-notify [OPTIONS...] [VARIABLE=VALUE...]

systemd-notify --exec [OPTIONS...] [VARIABLE=VALUE...] ; -- {CMDLINE...}

systemd-notify --fork [OPTIONS...] -- {CMDLINE...}

DESCRIPTION

systemd-notify may be called by service scripts to notify the invoking service manager about status changes. It can be used to send arbitrary information, encoded in an environment-block-like list of strings. Most importantly, it can be used for start-up completion notification.

This is mostly just a wrapper around sd_notify() and makes this functionality available to shell scripts. For details see sd_notify(3).

The command line may carry a list of environment variables to send as part of the status update.

Note that systemd will refuse reception of status updates from this command unless NotifyAccess= is appropriately set for the service unit this command is called from. See systemd.service(5) for details.

Note that sd_notify() notifications may be attributed to units correctly only if either the sending process is still around at the time the service manager processes the message, or if the sending process is explicitly runtime-tracked by the service manager. The latter is the

case if the service manager originally forked off the process, i.e. on all processes that match NotifyAccess=main or NotifyAccess=exec. Conversely, if an auxiliary process of the unit sends an sd_notify() message and immediately exits, the service manager might not be able to properly attribute the message to the unit, and thus will ignore it, even if NotifyAccess=all is set for it. To address this systemd-notify will wait until the notification message has been processed by the service manager. When --no-block is used, this synchronization for reception of notifications is disabled, and hence the aforementioned race may occur if the invoking process is not the service manager or spawned by the service manager.

systemd-notify will first attempt to invoke sd_notify() pretending to have the PID of the parent process of systemd-notify (i.e. the invoking process). This will only succeed when invoked with sufficient privileges. On failure, it will then fall back to invoking it under its own PID. This behaviour is useful in order that when the tool is invoked from a shell script the shell process ? and not the systemd-notify process ? appears as sender of the message, which in turn is helpful if the shell process is the main process of a service, due to the limitations of NotifyAccess=all. Use the --pid= switch to tweak this behaviour.

OPTIONS

The following options are understood:

--ready

Inform the invoking service manager about service start-up or configuration reload completion. This is equivalent to systemd-notify READY=1. For details about the semantics of this option see sd_notify(3).

--reloading

Inform the invoking service manager about the beginning of a configuration reload cycle. This is equivalent to systemd-notify RELOADING=1 (but implicitly also sets a MONOTONIC_USEC= field as required for Type=notify-reload services, see systemd.service(5) for details). For details about the semantics of this option see sd_notify(3).

Added in version 253.

--stopping

Inform the invoking service manager about the beginning of the shutdown phase of the service. This is equivalent to systemd-notify STOPPING=1. For details about the semantics of this option see sd_notify(3).

Added in version 253.

--pid=

Inform the service manager about the main PID of the service. Takes a PID as argument. If the argument is specified as "auto" or omitted, the PID of the process that invoked systemd-notify is used, except if that's the service manager. If the argument is specified as "self", the PID of the systemd-notify command itself is used, and if "parent" is specified the calling process' PID is used ? even if it is the service manager. --pid=auto is equivalent to systemd-notify --pid=$PID. For details about the semantics of this option see sd_notify(3).

systemd-notify will first attempt to invoke sd_notify() pretending to have the PID specified with --pid=. This will only succeed when invoked with sufficient privileges. On failure, it will then fall back to invoking it under its own PID. Effectively, this means that a privileged invocation of systemd-notify --pid= may circumvent NotifyAccess=main or NotifyAccess=exec restrictions enforced for a service.

If this switch is used in an unprivileged systemd-notify invocation from a process that shall become the new main process of a service ? and which is not the process forked off by the service manager (or the current main process) ?, then it is essential to set NotifyAccess=all in the service unit file, or otherwise the notification will be ignored for security reasons. See systemd.service(5) for details.

--uid=USER

Set the user ID to send the notification from. Takes a UNIX user name or numeric UID. When specified the notification message will be sent with the specified UID as sender, in place of the user the command was invoked as. This option requires sufficient privileges in order to be able manipulate the user identity of the process.

Added in version 237.

--status=

Send a free-form human-readable status string for the daemon to the service manager. This option takes the status string as argument. This is equivalent to systemd-notify STATUS=.... For details about the semantics of this option see sd_notify(3). This information is shown in systemctl(1)'s status output, among other places.

--booted

Returns 0 if the system was booted up with systemd, non-zero otherwise. If this option is passed, no message is sent. This option is hence unrelated to the other options. For

details about the semantics of this option, see sd_booted(3). An alternate way to check for this state is to call systemctl(1) with the is-system-running command. It will output "offline" if the system was not booted with systemd, though the return value has a different meaning.

--no-block

Do not synchronously wait for the requested operation to finish. Use of this option is only recommended when systemd-notify is spawned by the service manager, or when the invoking process is directly spawned by the service manager and has enough privileges to allow systemd-notify to send the notification on its behalf. Sending notifications with this option set is prone to race conditions in all other cases.

Added in version 246.

--exec

If specified systemd-notify will execute another command line after it completed its operation, replacing its own process. If used, the list of assignments to include in the message sent must be followed by a ";" character (as separate argument), followed by the command line to execute. This permits "chaining" of commands, i.e. issuing one operation, followed immediately by another, without changing PIDs.

Note that many shells interpret ";" as their own separator for command lines, hence when systemd-notify is invoked from a shell the semicolon must usually be escaped as "\;".

Added in version 254.

--fd=

Send a file descriptor along with the notification message. This is useful when invoked in services that have the FileDescriptorStoreMax= setting enabled, see systemd.service(5) for details. The specified file descriptor must be passed to systemd-notify when invoked. This option may be used multiple times to pass multiple file descriptors in a single notification message.

To use this functionality from a bash(1) shell, use an expression like the following:

```
systemd-notify --fd=4 --fd=5 4</some/file 5</some/other/file
```

Added in version 254.

--fdname=

Set a name to assign to the file descriptors passed via --fd= (see above). This controls the "FDNAME=" field. This setting may only be specified once, and applies to all file descriptors passed. Invoke this tool multiple times in case multiple file descriptors

with different file descriptor names shall be submitted.

Added in version 254.

--fork

Instead of sending a notification message, fork off a command line and wait until a "READY=1" message is received from it. In other words: this makes systemd-notify the receiver of notification messages instead of the sender, swapping roles. This is useful to quickly fork off a process that implements the sd_notify() protocol from a shell script. The invoked command line will have standard input and standard output connected to /dev/null, but standard error will be inherited from the invoking process. The numeric process ID is written to standard output by systemd-notify (unless --quiet is specified), which may be used to later terminate the forked off process.

Note that processes forked off like this will likely remain running after systemd-notify already returned, which hence will result in them being reparented to the closest process reaper process, i.e. typically the per-user or system service manager.

Note that this option should not be used to invoke full services ad-hoc, use systemd-run for that.

Also note that when invoked with this switch systemd-notify will exit successfully under two distinction conditions:

1. systemd-notify received a "READY=1" notification from the child it just forked off.
2. The child process exited cleanly (with exit status zero) before sending "READY=1".

Example use:

```
# PID=$(systemd-notify --fork -- mycommand)
...
kill "$PID"
unset PID
```

Added in version 258.

--quiet, -q

Turn off output of the numeric process ID when --fork is used.

Added in version 258.

-h, --help

Print a short help text and exit.

--version

Print a short version string and exit.

EXIT STATUS

On success, 0 is returned, a non-zero failure code otherwise.

EXAMPLE

Example 1. Start-up Notification and Status Updates

A simple shell daemon that sends start-up notifications after having set up its communication channel. During runtime it sends further status updates to the init system:

```
#!/bin/sh
mkfifo /tmp/waldo
systemd-notify --ready --status="Waiting for data..."
while : ; do
        read -r a < /tmp/waldo
        systemd-notify --status="Processing $a"
        # Do something with $a ...
        systemd-notify --status="Waiting for data..."
done
```

SEE ALSO

systemd(1), systemctl(1), systemd.unit(5), systemd.service(5), sd_notify(3), sd_booted(3)