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

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'systemd-run.1' command

$ man systemd-run.1

SYSTEMD-RUN(1) systemd-run SYSTEMD-RUN(1)

NAME

systemd-run - Run programs in transient scope units, service units, or path-, socket-, or timer-triggered service units

SYNOPSIS

systemd-run [OPTIONS...] COMMAND [ARGS...]

systemd-run [OPTIONS...] [PATH OPTIONS...] {COMMAND} [ARGS...]

systemd-run [OPTIONS...] [SOCKET OPTIONS...] {COMMAND} [ARGS...]

systemd-run [OPTIONS...] [TIMER OPTIONS...] {COMMAND} [ARGS...]

DESCRIPTION

systemd-run may be used to create and start a transient .service or .scope unit and run the specified COMMAND in it. It may also be used to create and start a transient .path, .socket, or .timer unit, that activates a .service unit when elapsing.

If a command is run as transient service unit, it will be started and managed by the service manager like any other service, and thus shows up in the output of systemctl list-units like any other unit. It will run in a clean and detached execution environment, with the service manager as its parent process. In this mode, systemd-run will start the service asynchronously in the background and return after the command has begun execution (unless --no-block or --wait are specified, see below).

If a command is run as transient scope unit, it will be executed by

systemd-run itself as parent process and will thus inherit the execution environment of the caller. However, the processes of the command are managed by the service manager similarly to normal services, and will show up in the output of systemctl list-units. Execution in this case is synchronous, and will return only when the command finishes. This mode is enabled via the --scope switch (see below).

If a command is run with path, socket, or timer options such as --on-calendar= (see below), a transient path, socket, or timer unit is created alongside the service unit for the specified command. Only the transient path, socket, or timer unit is started immediately, the transient service unit will be triggered by the path, socket, or timer unit. If the --unit= option is specified, the COMMAND may be omitted. In this case, systemd-run creates only a .path, .socket, or .timer unit that triggers the specified unit.

By default, services created with systemd-run default to the simple type, see the description of Type= in systemd.service(5) for details. Note that when this type is used, the service manager (and thus the systemd-run command) considers service start-up successful as soon as the fork() for the main service process succeeded, i.e. before the execve() is invoked, and thus even if the specified command cannot be started. Consider using the exec service type (i.e. --property=Type=exec) to ensure that systemd-run returns successfully only if the specified command line has been successfully started.

OPTIONS

The following options are understood:

--no-ask-password

Do not query the user for authentication for privileged operations.

--scope

Create a transient .scope unit instead of the default transient .service unit (see above).

--unit=, -u

Use this unit name instead of an automatically generated one.

--property=, -p

Sets a property on the scope or service unit that is created. This option takes an assignment in the same format as systemctl(1)'s set-property command.

--description=

Provide a description for the service, scope, path, socket, or timer unit. If not specified, the command itself will be used as a description. See Description= in systemd.unit(5).

--slice=

Make the new .service or .scope unit part of the specified slice, instead of system.slice (when running in --system mode) or the root slice (when running in --user mode).

--slice-inherit

Make the new .service or .scope unit part of the inherited slice. This option can be combined with --slice=.

An inherited slice is located within systemd-run slice. Example: if systemd-run slice is foo.slice, and the --slice= argument is bar, the unit will be placed under the foo-bar.slice.

-r, --remain-after-exit

After the service process has terminated, keep the service around until it is explicitly stopped. This is useful to collect runtime information about the service after it finished running. Also see RemainAfterExit= in systemd.service(5).

--send-sighup

When terminating the scope or service unit, send a SIGHUP immediately after SIGTERM. This is useful to indicate to shells and shell-like processes that the connection has been severed. Also see SendSIGHUP= in systemd.kill(5).

--service-type=

Sets the service type. Also see Type= in systemd.service(5). This option has no effect in conjunction with --scope. Defaults to simple.

--uid=, --gid=

Runs the service process under the specified UNIX user and group.

Also see User= and Group= in systemd.exec(5).

--nice=

Runs the service process with the specified nice level. Also see Nice= in systemd.exec(5).

--working-directory=

Runs the service process with the specified working directory. Also see WorkingDirectory= in systemd.exec(5).

--same-dir, -d

Similar to --working-directory=, but uses the current working directory of the caller for the service to execute.

-E NAME[=VALUE], --setenv=NAME[=VALUE]

Runs the service process with the specified environment variable set. This parameter may be used more than once to set multiple variables. When "=" and VALUE are omitted, the value of the variable with the same name in the program environment will be used.

Also see Environment= in systemd.exec(5).

--pty, -t

When invoking the command, the transient service connects its standard input, output and error to the terminal systemd-run is invoked on, via a pseudo TTY device. This allows running programs that expect interactive user input/output as services, such as interactive command shells.

Note that machinectl(1)'s shell command is usually a better alternative for requesting a new, interactive login session on the local host or a local container.

See below for details on how this switch combines with --pipe.

--pipe, -P

If specified, standard input, output, and error of the transient service are inherited from the systemd-run command itself. This allows systemd-run to be used within shell pipelines. Note that this mode is not suitable for interactive command shells and

similar, as the service process will not become a TTY controller when invoked on a terminal. Use --pty instead in that case. When both --pipe and --pty are used in combination the more appropriate option is automatically determined and used. Specifically, when invoked with standard input, output and error connected to a TTY --pty is used, and otherwise --pipe. When this option is used the original file descriptors systemd-run receives are passed to the service processes as-is. If the service runs with different privileges than systemd-run, this means the service might not be able to re-open the passed file descriptors, due to normal file descriptor access restrictions. If the invoked process is a shell script that uses the echo "hello" > /dev/stderr construct for writing messages to stderr, this might cause problems, as this only works if stderr can be re-opened. To mitigate this use the construct echo "hello" >&2 instead, which is mostly equivalent and avoids this pitfall.

--shell, -S

A shortcut for "--pty --same-dir --wait --collect --service-type=exec $SHELL", i.e. requests an interactive shell in the current working directory, running in service context, accessible with a single switch.

--quiet, -q

Suppresses additional informational output while running. This is particularly useful in combination with --pty when it will suppress the initial message explaining how to terminate the TTY connection.

--on-active=, --on-boot=, --on-startup=, --on-unit-active=, --on-unit-inactive=

Defines a monotonic timer relative to different starting points for starting the specified command. See OnActiveSec=, OnBootSec=, OnStartupSec=, OnUnitActiveSec= and OnUnitInactiveSec= in systemd.timer(5) for details. These options are shortcuts for --timer-property= with the relevant properties. These options may not be combined with --scope or --pty.

--on-calendar=

Defines a calendar timer for starting the specified command. See OnCalendar= in systemd.timer(5). This option is a shortcut for --timer-property=OnCalendar=. This option may not be combined with --scope or --pty.

--on-clock-change, --on-timezone-change

Defines a trigger based on system clock jumps or timezone changes for starting the specified command. See OnClockChange= and OnTimezoneChange= in systemd.timer(5). These options are shortcuts for --timer-property=OnClockChange=yes and --timer-property=OnTimezoneChange=yes. These options may not be combined with --scope or --pty.

--path-property=, --socket-property=, --timer-property=

Sets a property on the path, socket, or timer unit that is created. This option is similar to --property=, but applies to the transient path, socket, or timer unit rather than the transient service unit created. This option takes an assignment in the same format as systemctl(1)'s set-property command. These options may not be combined with --scope or --pty.

--no-block

Do not synchronously wait for the unit start operation to finish. If this option is not specified, the start request for the transient unit will be verified, enqueued and systemd-run will wait until the unit's start-up is completed. By passing this argument, it is only verified and enqueued. This option may not be combined with --wait.

--wait

Synchronously wait for the transient service to terminate. If this option is specified, the start request for the transient unit is verified, enqueued, and waited for. Subsequently the invoked unit is monitored, and it is waited until it is deactivated again (most likely because the specified command completed). On exit, terse information about the unit's runtime is shown, including total

runtime (as well as CPU usage, if --property=CPUAccounting=1 was set) and the exit code and status of the main process. This output may be suppressed with --quiet. This option may not be combined with --no-block, --scope or the various path, socket, or timer options.

-G, --collect

Unload the transient unit after it completed, even if it failed. Normally, without this option, all units that ran and failed are kept in memory until the user explicitly resets their failure state with systemctl reset-failed or an equivalent command. On the other hand, units that ran successfully are unloaded immediately. If this option is turned on the "garbage collection" of units is more aggressive, and unloads units regardless if they exited successfully or failed. This option is a shortcut for --property=CollectMode=inactive-or-failed, see the explanation for CollectMode= in systemd.unit(5) for further information.

--user

Talk to the service manager of the calling user, rather than the service manager of the system.

--system

Talk to the service manager of the system. This is the implied default.

-H, --host=

Execute the operation remotely. Specify a hostname, or a username and hostname separated by "@", to connect to. The hostname may optionally be suffixed by a port ssh is listening on, separated by ":", and then a container name, separated by "/", which connects directly to a specific container on the specified host. This will use SSH to talk to the remote machine manager instance. Container names may be enumerated with machinectl -H HOST. Put IPv6 addresses in brackets.

-M, --machine=

Execute operation on a local container. Specify a container name to

connect to, optionally prefixed by a user name to connect as and a separating "@" character. If the special string ".host" is used in place of the container name, a connection to the local system is made (which is useful to connect to a specific user's user bus: "--user --machine=lennart@.host"). If the "@" syntax is not used, the connection is made as root user. If the "@" syntax is used either the left hand side or the right hand side may be omitted (but not both) in which case the local user name and ".host" are implied.

-h, --help

Print a short help text and exit.

--version

Print a short version string and exit.

All command line arguments after the first non-option argument become part of the command line of the launched process.

EXIT STATUS

On success, 0 is returned. If systemd-run failed to start the service, a non-zero return value will be returned. If systemd-run waits for the service to terminate, the return value will be propagated from the service. 0 will be returned on success, including all the cases where systemd considers a service to have exited cleanly, see the discussion of SuccessExitStatus= in systemd.service(5).

EXAMPLES

Example 1. Logging environment variables provided by systemd to services

```
# systemd-run env
Running as unit: run-19945.service
# journalctl -u run-19945.service
Sep 08 07:37:21 bupkis systemd[1]: Starting /usr/bin/env...
Sep 08 07:37:21 bupkis systemd[1]: Started /usr/bin/env.
Sep 08 07:37:21 bupkis env[19948]: PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
Sep 08 07:37:21 bupkis env[19948]: LANG=en_US.UTF-8
Sep 08 07:37:21 bupkis env[19948]: BOOT_IMAGE=/vmlinuz-3.11.0-0.rc5.git6.2.fc20.x86_64
```

Example 2. Limiting resources available to a command

```
# systemd-run -p IOWeight=10 updatedb
```

This command invokes the updatedb(8) tool, but lowers the block I/O weight for it to 10. See systemd.resource-control(5) for more information on the IOWeight= property.

Example 3. Running commands at a specified time

The following command will touch a file after 30 seconds.

```
# date; systemd-run --on-active=30 --timer-property=AccuracySec=100ms /bin/touch /tmp/foo
Mon Dec  8 20:44:24 KST 2014
Running as unit: run-71.timer
Will run service as unit: run-71.service
# journalctl -b -u run-71.timer
-- Journal begins at Fri 2014-12-05 19:09:21 KST, ends at Mon 2014-12-08 20:44:54 KST. --
Dec 08 20:44:38 container systemd[1]: Starting /bin/touch /tmp/foo.
Dec 08 20:44:38 container systemd[1]: Started /bin/touch /tmp/foo.
# journalctl -b -u run-71.service
-- Journal begins at Fri 2014-12-05 19:09:21 KST, ends at Mon 2014-12-08 20:44:54 KST. --
Dec 08 20:44:48 container systemd[1]: Starting /bin/touch /tmp/foo...
Dec 08 20:44:48 container systemd[1]: Started /bin/touch /tmp/foo.
```

Example 4. Allowing access to the tty

The following command invokes bash(1) as a service passing its standard input, output and error to the calling TTY.

```
# systemd-run -t --send-sighup bash
```

Example 5. Start screen as a user service

```
$ systemd-run --scope --user screen
Running scope as unit run-r14b0047ab6df45bfb45e7786cc839e76.scope.
$ screen -ls
There is a screen on:
        492..laptop     (Detached)
1 Socket in /var/run/screen/S-fatima.
```

This starts the screen process as a child of the systemd --user process that was started by user@.service, in a scope unit. A systemd.scope(5) unit is used instead of a systemd.service(5) unit, because screen will

exit when detaching from the terminal, and a service unit would be terminated. Running screen as a user unit has the advantage that it is not part of the session scope. If KillUserProcesses=yes is configured in logind.conf(5), the default, the session scope will be terminated when the user logs out of that session.

The user@.service is started automatically when the user first logs in, and stays around as long as at least one login session is open. After the user logs out of the last session, user@.service and all services underneath it are terminated. This behavior is the default, when "lingering" is not enabled for that user. Enabling lingering means that user@.service is started automatically during boot, even if the user is not logged in, and that the service is not terminated when the user logs out.

Enabling lingering allows the user to run processes without being logged in, for example to allow screen to persist after the user logs out, even if the session scope is terminated. In the default configuration, users can enable lingering for themselves:

```
$ loginctl enable-linger
```

Example 6. Return value

```
$ systemd-run --user --wait true
$ systemd-run --user --wait -p SuccessExitStatus=11 bash -c 'exit 11'
$ systemd-run --user --wait -p SuccessExitStatus=SIGUSR1 bash -c 'kill -SIGUSR1 $$$$'
```

Those three invocations will succeed, i.e. terminate with an exit code of 0.

SEE ALSO

systemd(1), systemctl(1), systemd.unit(5), systemd.service(5), systemd.scope(5), systemd.slice(5), systemd.exec(5), systemd.resource-control(5), systemd.timer(5), systemd-mount(1), machinectl(1)