

systemd Unit File Cheatsheet

By Dejan Panovski • Updated on Sep 30, 2026

systemd unit file directives at a glance: [Unit] dependencies, service types, Exec commands, restart policy, users and environment, resource limits, hardening, timers, and drop-in overrides.

A unit file is the plain-text configuration systemd reads to decide how to start, stop, and supervise a service, timer, or other unit. This cheatsheet lists the directives you reach for most often when you write a `.service` or `.timer` file, plus the commands that install, override, and verify it.

File Locations

systemd reads units from several directories. `/etc` overrides `/run`, which overrides `/usr/lib`.

<code>/etc/systemd/system/</code>	Units you write or override as the administrator
<code>/run/systemd/system/</code>	Runtime units, gone after a reboot
<code>/usr/lib/systemd/system/</code>	Units installed by packages; do not edit these
<code>/etc/systemd/system/name.service.d/*.conf</code>	Drop-in overrides for one unit
<code>~/.config/systemd/user/</code>	Per-user units, managed with <code>systemctl --user</code>
<code>/etc/systemd/user/</code>	User units for every account on the system
<code>systemctl cat name.service</code>	Print the unit and every drop-in that applies to it

Unit Types

The file extension decides what kind of unit systemd creates.

<code>.service</code>	A process or daemon
<code>.timer</code>	Schedule that activates another unit
<code>.socket</code>	Listening socket that starts a service on demand
<code>.target</code>	Group of units, used as a boot milestone
<code>.path</code>	Watches a file or directory and activates a unit on change
<code>.mount</code>	Filesystem mount point
<code>.slice</code>	Resource control group for other units

[Unit] Section

Description, ordering, and dependencies.

Description=My API server	Name shown by <code>systemctl status</code>
Documentation=man:nginx(8) https://nginx.org/	Documentation links
After=network-online.target	Start after the named unit; ordering only
Before=nginx.service	Start before the named unit
Wants=network-online.target	Pull in a unit, keep running if it fails
Requires=postgresql.service	Pull in a unit, stop if it is explicitly stopped
BindsTo=dev-sdb1.device	Stop as soon as the named unit stops
PartOf=app.target	Stop and restart together with the named unit
Conflicts=shutdown.target	Stop the named unit when this one starts, and the reverse
ConditionPathExists=/etc/app.conf	Skip the start quietly when the check fails
AssertPathExists=/etc/app.conf	Fail the start when the check fails
StartLimitIntervalSec=60	Window for counting start attempts
StartLimitBurst=5	Starts allowed inside that window

`Wants=` and `Requires=` do not set start order. Pair them with `After=` when one unit must be up first.

Service Types

`Type=` tells systemd when the service counts as started.

Type=simple	Started as soon as the process is forked; the default
Type=exec	Started once the binary has been executed successfully
Type=forking	Started when the parent exits; the daemon forks itself
PIDFile=/run/app.pid	PID file to track for a forking daemon
Type=oneshot	Runs to completion; later units wait for it to exit
RemainAfterExit=yes	Keep a oneshot unit shown as active after it exits
Type=notify	Started when the process sends <code>READY=1</code> via <code>sd_notify</code>
Type=notify-reload	Like <code>notify</code> , and reloads by sending <code>SIGHUP</code>
Type=dbus	Started when the name in <code>BusName=</code> appears on D-Bus
Type=idle	Like <code>simple</code> , but waits until other boot jobs finish

Exec Commands

Commands systemd runs at each stage of the service lifecycle.

<code>ExecStart=/usr/bin/app --port 8080</code>	Main command; use an absolute path
<code>ExecStartPre=/usr/bin/app --check-config</code>	Run before <code>ExecStart=</code> ; a failure stops the start
<code>ExecStartPost=/usr/bin/notify-ready</code>	Run after the service has started
<code>ExecReload=/bin/kill -HUP \$MAINPID</code>	Run by <code>systemctl reload</code>
<code>ExecStop=/usr/bin/app --shutdown</code>	Run by <code>systemctl stop</code> before the signal is sent
<code>ExecStopPost=/usr/bin/cleanup</code>	Run after the service has stopped or failed
<code>ExecStartPre=-/usr/bin/mkdir /tmp/app</code>	- prefix: ignore a non-zero exit code
<code>ExecStartPre=+/usr/bin/chown app /data</code>	+ prefix: run with full privileges, ignoring <code>User=</code> and sandboxing
<code>ExecStart=:/usr/bin/echo \$HOME</code>	: prefix: pass \$ through without variable expansion

Commands are not run through a shell, so pipes, `>`, and `&&` do not work. Wrap them in `/bin/sh -c '...'` or move them into a script.

Restart and Timeouts

Control what happens when the process exits or hangs.

<code>Restart=on-failure</code>	Restart on non-zero exit, signal, timeout, or watchdog
<code>Restart=always</code>	Restart no matter how the process exited
<code>Restart=on-abnormal</code>	Restart on signal, timeout, or watchdog, not on exit codes
<code>Restart=no</code>	Never restart; the default
<code>RestartSec=5</code>	Wait five seconds before restarting
<code>RestartSteps=5</code>	Grow the delay over five restarts
<code>RestartMaxDelaySec=60</code>	Upper limit for the growing delay
<code>TimeoutStartSec=30</code>	Fail the start after 30 seconds
<code>TimeoutStopSec=30</code>	Escalate to <code>SIGKILL</code> after 30 seconds
<code>SuccessExitStatus=143</code>	Treat an extra exit code as a clean exit
<code>RestartPreventExitStatus=2</code>	Never restart after this exit code
<code>KillMode=mixed</code>	<code>SIGTERM</code> to the main process, <code>SIGKILL</code> to the rest of the group
<code>WatchdogSec=30</code>	Mark the service failed when keep-alive pings stop

A unit that restarts more than `StartLimitBurst=` times inside `StartLimitIntervalSec=` is refused further starts until the interval passes or you run `systemctl reset-failed`.

User, Environment, and Directories

Who the process runs as and what it sees.

User=app	Run as this user instead of root
Group=app	Run with this primary group
DynamicUser=yes	Allocate a temporary user for each run
WorkingDirectory=/opt/app	Directory the process starts in
Environment="NODE_ENV=production" "PORT=8080"	Set variables inline
EnvironmentFile=/etc/app/app.env	Read KEY=value lines from a file
EnvironmentFile=-/etc/default/app	Same, but skip the file when it is missing
UMask=0027	Default permission mask for new files
StateDirectory=app	Create /var/lib/app owned by the service user
RuntimeDirectory=app	Create /run/app, removed when the service stops
LogsDirectory=app	Create /var/log/app
ConfigurationDirectory=app	Create /etc/app
StandardOutput=journal	Send stdout to the journal; the default
SyslogIdentifier=app	Tag used in journalctl output

Resource Limits

Cap what one service can consume.

LimitNOFILE=65536	Maximum open files
LimitNPROC=4096	Maximum processes for the service user
MemoryMax=512M	Hard memory limit; the OOM killer acts above it
MemoryHigh=400M	Throttle and reclaim memory above this level
CPUQuota=50%	Half of one CPU; 200% means two CPUs
CPUWeight=50	Relative CPU share; the default is 100
IOWeight=50	Relative disk I/O share; the default is 100
TasksMax=100	Maximum threads and processes
Nice=10	Scheduling priority, from -20 to 19
systemctl set-property app.service MemoryMax=1G	Change a limit at runtime and save it as a drop-in

`ulimit` values set in a shell do not apply to services. Use the `Limit*` directives instead.

Security Hardening

Sandboxing options that most services tolerate.

<code>NoNewPrivileges=yes</code>	Block privilege gain through <code>setuid</code> binaries
<code>ProtectSystem=strict</code>	Mount the whole file system read-only
<code>ProtectSystem=full</code>	Mount <code>/usr</code> , <code>/boot</code> , and <code>/etc</code> read-only
<code>ReadWritePaths=/var/lib/app</code>	Keep one path writable under <code>ProtectSystem=</code>
<code>ProtectHome=yes</code>	Hide <code>/home</code> , <code>/root</code> , and <code>/run/user</code>
<code>ProtectHome=read-only</code>	Show home directories, but read-only
<code>PrivateTmp=yes</code>	Give the service its own <code>/tmp</code> and <code>/var/tmp</code>
<code>PrivateDevices=yes</code>	Hide physical devices under <code>/dev</code>
<code>ProtectKernelTunables=yes</code>	Make <code>/proc/sys</code> and <code>/sys</code> read-only
<code>ProtectKernelModules=yes</code>	Block kernel module loading
<code>AmbientCapabilities=CAP_NET_BIND_SERVICE</code>	Let a non-root user bind ports below 1024
<code>CapabilityBoundingSet=CAP_NET_BIND_SERVICE</code>	Drop every other capability
<code>RestrictAddressFamilies=AF_INET AF_INET6 AF_UNIX</code>	Allow only IP and Unix sockets
<code>SystemCallFilter=@system-service</code>	Allow the system calls a typical service needs
<code>systemd-analyze security app.service</code>	Score the unit's exposure and list what is missing

[Install] Section

Read only by `systemctl enable` and `disable`.

<code>WantedBy=multi-user.target</code>	Start at boot on servers and desktops
<code>WantedBy=graphical.target</code>	Start only when the graphical session target is reached
<code>WantedBy=default.target</code>	Start with the user session, for <code>--user</code> units
<code>WantedBy=timers.target</code>	Activate a timer at boot
<code>RequiredBy=app.target</code>	Hard dependency created on enable
<code>Alias=api.service</code>	Extra name created as a symlink on enable
<code>Also=app.socket</code>	Enable or disable another unit together with this one

A unit without an `[Install]` section can be started by hand but not enabled.

Timer Units

A `name.timer` activates `name.service` by default.

<code>OnCalendar=daily</code>	Every day at midnight
<code>OnCalendar=*-*-* 02:30:00</code>	Every day at 02:30
<code>OnCalendar=Mon..Fri 09:00</code>	Weekdays at 09:00
<code>OnCalendar=*:0/15</code>	Every 15 minutes
<code>OnCalendar=*-*-01 00:00</code>	First day of every month
<code>OnBootSec=5min</code>	Five minutes after boot
<code>OnUnitActiveSec=1h</code>	One hour after the unit last started
<code>Persistent=true</code>	Run a missed calendar event after downtime
<code>RandomizedDelaySec=10m</code>	Spread the start by up to 10 minutes
<code>AccuracySec=1s</code>	Tighten the default one-minute accuracy
<code>Unit=backup.service</code>	Activate a unit with a different name
<code>systemd-analyze calendar "Mon..Fri 09:00"</code>	Check an expression and show the next run
<code>systemctl list-timers</code>	Show active timers and their next run

Overrides and Drop-ins

Change a packaged unit without editing the file under `/usr/lib`.

<code>sudo systemctl edit nginx.service</code>	Create or edit a drop-in override
<code>sudo systemctl edit --full nginx.service</code>	Copy the whole unit to <code>/etc</code> and edit it
<code>sudo systemctl revert nginx.service</code>	Remove overrides and return to the vendor unit
<code>systemctl cat nginx.service</code>	Show the unit with every drop-in applied
<code>systemd-delta</code>	List units that are overridden or extended
<code>ExecStart=</code>	Empty line in a drop-in clears the list before a new <code>ExecStart=</code>
<code>sudo systemctl daemon-reload</code>	Reload unit files after a manual change

`systemctl edit` reloads the configuration on save. Edits made with a normal editor need `daemon-reload`.

Specifiers

Placeholders systemd expands inside unit files.

<code>%n</code>	Full unit name, such as <code>app@web.service</code>
<code>%N</code>	Unit name without the type suffix
<code>%i</code>	Instance name in a template unit, the part after <code>@</code>
<code>%H</code>	Host name
<code>%h</code>	Home directory of the user running the manager
<code>%u</code>	User name of the manager
<code>%t</code>	Runtime directory: <code>/run</code> or <code>\$XDG_RUNTIME_DIR</code>
<code>%%</code>	A literal percent sign

A template file such as `app@.service` is started as `app@web.service`, with `%i` set to `web`.

Install and Verify

Put a new unit in place and confirm it loads.

<code>sudo cp app.service /etc/systemd/system/</code>	Install the unit file
<code>systemd-analyze verify /etc/systemd/system/app.service</code>	Check syntax and unknown directives
<code>sudo systemctl daemon-reload</code>	Make systemd read the new file
<code>sudo systemctl enable --now app.service</code>	Enable at boot and start now
<code><u>systemctl status app.service</u></code>	Show state and the latest log lines
<code><u>journalctl -u app.service -f</u></code>	Follow the service log
<code>systemctl show app.service -p Restart,User</code>	Print the effective value of directives
<code>systemctl list-dependencies app.service</code>	Show the dependency tree
<code>systemctl --user daemon-reload</code>	Reload units in <code>~/.config/systemd/user/</code>
<code>sudo loginctl enable-linger username</code>	Keep user units running after logout

Related Guides

Deeper reading on services and systemd.

[How to Create a systemd Service File](#)

Write, install, and manage a service step by step

[systemd Service File Generator](#)

Build a `.service` file in the browser

[systemctl Command in Linux](#)

Start, stop, enable, and inspect units

[How to List Services with systemctl](#)

Filter units by state and type

[journalctl Command in Linux](#)

Read and filter service logs

[systemctl cheatsheet](#)

Quick reference for `systemctl`

[journalctl cheatsheet](#)

Quick reference for `journalctl`