Systemd Cheatsheet

Systemd Cheatsheet

systemctl — Service Management

```
# Start, stop, restart a service
sudo systemctl start nginx
sudo systemctl stop nginx
sudo systemctl restart nginx

# Reload configuration (graceful, no downtime — service must support SIGHUP)
sudo systemctl reload nginx

# Conditionally restart only if the service is running
sudo systemctl try-restart nginx

# Enable/disable a service (start/stop at boot)
sudo systemctl enable nginx
sudo systemctl disable nginx

# Enable and start immediately
sudo systemctl enable --now nginx
# Disable and stop immediately
sudo systemctl disable --now nginx

# Check service status
sudo systemctl status nginx

# Check whether a service is active/enabled
systemctl is-active nginx        # prints: active / inactive / failed
systemctl is-enabled nginx       # prints: enabled / disabled / static / masked
systemctl is-failed nginx        # prints: failed / active / inactive

# Mask a service (makes it impossible to start, even manually)
sudo systemctl mask sleep.target
# Unmask
sudo systemctl unmask sleep.target
```

systemctl — Listing Units

```
# List all units (services, sockets, timers, targets, etc.)
systemctl list-units

# List only active units
systemctl list-units --state=active

# List failed units
systemctl list-units --state=failed

# List all unit files on disk (including inactive)
systemctl list-unit-files

# List services specifically
systemctl list-units --type=service

# List timers
systemctl list-timers --all

# List sockets
systemctl list-sockets

# Show dependencies of a unit
systemctl list-dependencies nginx
systemctl list-dependencies --reverse nginx   # reverse (what depends on it)

# Show a unit's full configuration (resolved, with drop-ins merged)
systemctl cat nginx
```

systemctl — System State

```
# Check overall system state
systemctl status                  # summary of system
systemctl is-system-running       # prints: running / degraded / maintenance

# Halt, power-off, reboot
sudo systemctl halt
sudo systemctl poweroff
sudo systemctl reboot

# Suspend / hibernate
sudo systemctl suspend
sudo systemctl hibernate
sudo systemctl hybrid-sleep       # suspend + hibernate

# Enter rescue / emergency mode
sudo systemctl rescue
sudo systemctl emergency

# Change default target (runlevel)
sudo systemctl set-default multi-user.target
sudo systemctl set-default graphical.target

# Get current default target
systemctl get-default
```

Unit File Structure

All unit files live under `/etc/systemd/system/` (admin-managed) or `/lib/systemd/system/` (package-managed). File name = unit name, e.g. `myapp.service`.

```
[Unit]
# Description shown in logs and status output
Description=My Application Service
Documentation=https://docs.example.com/myapp

# --- Dependency ordering ---
# After: start this unit AFTER the listed units have started
After=network-online.target docker.service
# Before: start this unit BEFORE the listed units
Before=nginx.service
# Requires: hard dependency — if the listed unit fails, this unit also fails
Requires=docker.service
# Wants: soft dependency — start the listed unit if possible, but don't fail if it can't
Wants=redis.service
# PartOf: restart this unit when the parent restarts
PartOf=docker.service
# Conflicts: this unit cannot run alongside the listed unit
Conflicts=shutdown.target

[Service]
# --- Execution ---
# ExecStart: the main process command (required for Type=simple)
ExecStart=/usr/bin/myapp --config /etc/myapp/config.toml
# ExecStartPre: run before ExecStart (must succeed or the service fails)
ExecStartPre=/usr/bin/myapp --validate-config
# ExecStartPost: run after the main process has started
ExecStartPost=/usr/bin/myapp --notify-master
# ExecReload: command to reload config (triggered by systemctl reload)
ExecReload=/bin/kill -HUP $MAINPID
# ExecStop: command to stop the service (default: SIGTERM)
ExecStop=/usr/bin/myapp --shutdown
# ExecStopPost: cleanup after the service has stopped
ExecStopPost=/usr/bin/rm -f /run/myapp.pid

# --- Service type ---
Type=simple           # default: service is considered started once ExecStart forks
Type=forking          # service forks into background; systemd tracks PID via PIDFile
Type=notify           # service sends sd_notify() when ready
Type=oneshot          # service runs once and exits; use RemainAfterExit=yes
Type=dbus             # service is ready when a specified bus name appears
Type=idle             # like simple, but delays start until all jobs are dispatched

PIDFile=/run/myapp.pid   # required for Type=forking

# --- Restart policy ---
Restart=on-failure       # restart on non-zero exit or signal
Restart=always           # always restart (even on clean exit)
Restart=on-abnormal      # on signal, timeout, or watchdog
Restart=no               # never restart (default)

# Delay between restart attempts (default: 100ms)
RestartSec=5s

# How many times to attempt restart within a time window
StartLimitBurst=5        # max attempts
StartLimitIntervalSec=60 # within this window
```

```
# --- User and permissions ---
User=myapp
Group=myapp
WorkingDirectory=/opt/myapp
# SupplementaryGroups=docker  # additional groups

# --- Environment ---
Environment="NODE_ENV=production"
Environment="DB_HOST=localhost"
Environment="DB_PORT=5432"
EnvironmentFile=/etc/myapp/env    # load KEY=VALUE lines from a file
EnvironmentFile=-/etc/myapp/env.local  # dash prefix = don't fail if missing

# --- Resource limits ---
LimitNOFILE=65536           # max open file descriptors
LimitNPROC=4096             # max processes
MemoryMax=512M              # memory limit (kill if exceeded)
MemoryHigh=384M             # soft memory limit (throttle)
CPUQuota=50%                # CPU limit (percentage of a single core)
TasksMax=100                # max number of threads/processes
TimeoutStartSec=30          # max time to start before systemd kills it
TimeoutStopSec=10           # max time to stop
TimeoutSec=60               # shorthand for both start and stop

# --- Security hardening ---
# Sandboxing
ProtectSystem=strict        # make /usr and /boot read-only
ProtectHome=true            # make /home, /root, /run/user inaccessible
PrivateTmp=yes              # mount a private /tmp
PrivateDevices=yes          # mount a private /dev (no physical devices)
PrivateNetwork=yes          # isolate network namespace (no network access)
NoNewPrivileges=yes         # prevent setuid/setgid privilege escalation
ReadOnlyPaths=/etc/myapp    # explicitly allow read-only paths
ReadWritePaths=/var/lib/myapp  # explicitly allow read-write paths
ProtectKernelTunables=yes   # prevent modifying kernel tunables
ProtectControlGroups=yes    # prevent modifying cgroups
ProtectKernelModules=yes    # prevent loading/unloading kernel modules
SystemCallFilter=@system-service  # whitelist allowed syscalls
CapabilityBoundingSet=CAP_NET_BIND_SERVICE  # restrict Linux capabilities
AmbientCapabilities=CAP_NET_BIND_SERVICE   # grant without full root

# --- Logging ---
StandardOutput=journal       # log stdout to journald (default)
StandardError=journal        # log stderr to journald (default)
SyslogIdentifier=myapp       # identifier for syslog/journal filtering

# --- Notify systemd when ready ---
NotifyAccess=all             # allow all processes to send readiness notifications
WatchdogSec=30               # expect a keepalive ping every 30s

# --- Misc ---
RemainAfterExit=yes          # mark service as active after exit (for Type=oneshot)

[Install]
# WantedBy: the target that pulls in this unit when enabled
WantedBy=multi-user.target
# Also= also enable these units when this one is enabled
Also=myapp-logrotate.timer
```

```
# Alias= create symlinks with these names
Alias=myapp.service
```

Real-World Service Unit File

```
# /etc/systemd/system/myapp.service
[Unit]
Description=MyApp Web Server
After=network-online.target postgresql.service
Wants=postgresql.service
StartLimitIntervalSec=300
StartLimitBurst=5

[Service]
Type=simple
User=myapp
Group=myapp
WorkingDirectory=/opt/myapp
ExecStartPre=/usr/bin/myapp migrate --config /etc/myapp/config.toml
ExecStart=/usr/bin/myapp serve --config /etc/myapp/config.toml --port 8080
ExecReload=/bin/kill -HUP $MAINPID
Restart=on-failure
RestartSec=5s
TimeoutStartSec=30
TimeoutStopSec=10

EnvironmentFile=/etc/myapp/environment
Environment="RUST_LOG=info"

LimitNOFILE=65536
MemoryMax=1G
TasksMax=512

ProtectSystem=strict
ProtectHome=true
PrivateTmp=yes
NoNewPrivileges=yes
ReadOnlyPaths=/etc/myapp
ReadWritePaths=/var/lib/myapp /var/log/myapp

StandardOutput=journal
StandardError=journal
SyslogIdentifier=myapp

[Install]
WantedBy=multi-user.target
```

Drop-In Directories

Drop-ins let you override or extend a unit's settings without modifying the original file. Useful for vendor units you don't control.

```
# Create a drop-in for nginx
sudo systemctl edit nginx
# This creates: /etc/systemd/system/nginx.service.d/override.conf
```

```
# Create a drop-in with a custom name
sudo systemctl edit nginx --force --full
```

```
# Directory structure
/etc/systemd/system/
  nginx.service             # your custom unit (takes priority)
  nginx.service.d/          # drop-in directory
    override.conf           # merged on top of the vendor unit
    custom-limits.conf      # additional drop-in file
/lib/systemd/system/
  nginx.service             # vendor/package unit
```

```
# /etc/systemd/system/nginx.service.d/custom-limits.conf
[Service]
LimitNOFILE=100000
MemoryMax=2G
Environment="NGINX_WORKER_PROCESSES=auto"
```

```
# View the fully resolved (merged) unit configuration
systemctl cat nginx

# List active drop-in directories
systemctl show nginx | grep DropInPaths
```

Timer Units

Systemd timers replace cron for most scheduling use cases. They support monotonic timers (relative to boot, not wall clock).

```
# List all timers
systemctl list-timers --all

# Show a timer's status
systemctl status myapp-backup.timer
```

```
# /etc/systemd/system/myapp-backup.timer
[Unit]
Description=Run MyApp backup daily

[Timer]
# Wall clock: run at a specific time
OnCalendar=daily
OnCalendar=*-*-* 03:00:00         # every day at 3 AM
OnCalendar=Mon *-*-* 02:30:00     # every Monday at 2:30 AM
OnCalendar=hourly                 # every hour
OnCalendar=*:0/15                 # every 15 minutes
OnCalendar=Mon,Fri *-*-* 01,13:00 # Mon and Fri at 1 AM and 1 PM
OnCalendar=monthly                # on the 1st of every month at midnight
OnCalendar=quarterly              # Jan 1, Apr 1, Jul 1, Oct 1 at midnight
OnCalendar=2026-12-25 00:00:00    # one-shot on a specific date

# Monotonic timers (relative to an event)
OnBootSec=5min                    # 5 minutes after boot
OnStartupSec=10min                # 10 minutes after systemd starts
```

```
OnUnitActiveSec=1d               # 1 day after the timer was last activated
OnUnitActiveSec=1h               # 1 hour after last activation

# Timer behavior
Persistent=true     # if the system was off at the scheduled time, run it at next boot
AccuracySec=1min    # allow up to 1 minute drift (reduces wake-ups, saves power)
RandomizedDelaySec=5m  # random delay up to 5 minutes (avoid thundering herd)
WakeSystem=false     # don't wake the system from suspend

# Only useful for one-shot timers:
Unit=myapp-backup.service

[Install]
WantedBy=timers.target
```

```
# Enable/start a timer (not the service — the timer activates it)
sudo systemctl enable --now myapp-backup.timer

# Check when a timer will next fire
systemctl list-timers myapp-backup.timer
```

Timers vs Cron

Feature	Systemd Timers	Cron
Wall clock scheduling	Yes (`OnCalendar`)	Yes
Boot-relative scheduling	Yes (`OnBootSec`)	No
Catch-up missed runs	Yes (`Persistent=yes`)	No (anacron partial)
Logging	Integrated with journalctl	Usually email or syslog
Dependencies	Full unit dependency chain	No
Per-service timezone	Yes	No
Randomized delay	Built-in	Needs wrapper script
Resource limits	Via service unit	No
Millisecond precision	Yes	Minute precision
Last status tracking	Yes (`systemctl status`)	No

Rule of thumb: Use systemd timers for anything running on a systemd-based system. Use cron only for compatibility with non-systemd environments.

Target Units (Runlevels)

Targets group units together, similar to traditional SysV runlevels.

```
# View current target
systemctl get-default

# Change default target
sudo systemctl set-default multi-user.target

# List all available targets
systemctl list-unit-files --type=target
```

```
# Isolate a target (switch to it, stopping everything not in its dependency tree)
sudo systemctl isolate rescue.target
```

SysV Runlevel	Systemd Target	Purpose
0	`poweroff.target`	Halt / power off
1	`rescue.target`	Single-user / rescue mode
2, 3, 4	`multi-user.target`	Multi-user, no GUI
5	`graphical.target`	Multi-user with GUI
6	`reboot.target`	Reboot
—	`emergency.target`	Emergency shell (minimal)
—	`network.target`	Network is up
—	`network-online.target`	Network is fully routable
—	`basic.target`	Basic system initialized
—	`timers.target`	All timers
—	`graphical.target`	Display manager active

```
# Common targets to know
systemctl list-dependencies multi-user.target   # see what starts in multi-user mode
systemctl list-dependencies graphical.target
```

Socket Units

Socket activation lets systemd listen on a port/socket and start the service on-demand when a connection arrives.

```
# /etc/systemd/system/myapp.socket
[Unit]
Description=MyApp Socket

[Socket]
ListenStream=8080
Accept=no              # no = pass the listening socket to the service
                        # yes = systemd accepts, spawns service per connection
Service=myapp.service

[Install]
WantedBy=sockets.target
```

```
# Enable the socket (not the service)
sudo systemctl enable --now myapp.socket

# Check which sockets are active
systemctl list-sockets

# Verify systemd is listening
ss -tlnp | grep 8080
```

Mount Units

Systemd can manage mount points, including automount (lazy mounting on first access).

```
# /etc/systemd/system/mnt-backup.mount
[Unit]
Description=Mount backup drive
Requires=network-online.target
After=network-online.target

[Mount]
What=192.168.1.100:/backups
Where=/mnt/backups
Type=nfs
Options=ro,noauto,_netdev

[Install]
WantedBy=multi-user.target
```

```
# /etc/systemd/system/mnt-backup.automount
[Unit]
Description=Automount backup drive

[Automount]
Where=/mnt/backups
TimeoutIdleSec=300    # unmount after 5 minutes of inactivity

[Install]
WantedBy=multi-user.target
```

```
# Enable automount
sudo systemctl enable --now mnt-backup.automount

# List mount units
systemctl list-units --type=mount
```

journalctl — Log Querying

```
# Show all logs (newest first)
journalctl

# Follow logs (like tail -f)
journalctl -f

# Show logs for a specific service
journalctl -u nginx
journalctl -u nginx -f          # follow nginx logs

# Show logs for a specific PID
journalctl _PID=1234

# Show logs for a specific user
journalctl _UID=1000

# Time-based filtering
journalctl --since "2026-04-20"
journalctl --since "2026-04-20 09:00:00"
journalctl --since "1 hour ago"
```

```
journalctl --since "yesterday"
journalctl --until "2026-04-20 18:00:00"
journalctl --since "2026-04-19" --until "2026-04-20"

# Combine filters
journalctl -u nginx --since "2026-04-20 09:00" --until "2026-04-20 12:00"

# Show logs from the current boot only
journalctl -b

# Show logs from a specific boot
journalctl -b -1          # previous boot
journalctl -b -2          # two boots ago
journalctl --list-boots   # list all recorded boots with timestamps

# Priority filtering (0=emerg, 1=alert, 2=crit, 3=err, 4=warning, 5=notice, 6=info, 7=debug)
journalctl -p err         # errors and above
journalctl -p warning     # warnings and above
journalctl -p 3           # same as err

# Show kernel messages only
journalctl -k

# Output format
journalctl -o json        # structured JSON (one object per line)
journalctl -o json-pretty # pretty-printed JSON
journalctl -o verbose     # all fields visible
journalctl -o short-precise  # include microsecond timestamps
journalctl -o short-iso   # ISO 8601 timestamps

# Show only the latest N entries
journalctl -n 50
journalctl -n 100 --no-pager   # first 100 lines, no pager

# Filter by executable or systemd unit
journalctl /usr/sbin/sshd
journalctl _SYSTEMD_UNIT=ssh.service

# Grep through log messages
journalctl -u nginx | grep "error"
journalctl -u nginx -g "connection.*refused"   # use journal's built-in grep

# Disk usage
journalctl --disk-usage

# Vacuum old logs (keep last 500 MB)
sudo journalctl --vacuum-size=500M

# Vacuum by time
sudo journalctl --vacuum-time=2weeks
sudo journalctl --vacuum-time=1month

# Vacuum by number of files
sudo journalctl --vacuum-files=10
```

Troubleshooting

```
# Check why a service failed
systemctl status nginx
journalctl -u nginx -n 50 --no-pager

# Check if a service's unit file has syntax errors
systemd-analyze verify /etc/systemd/system/myapp.service

# Analyze boot time
systemd-analyze
systemd-analyze blame          # per-unit boot time
systemd-analyze critical-chain   # dependency chain that took longest
systemd-analyze critical-chain nginx.service

# Check which processes a service has running
systemctl show nginx --property=MainPID --property=ControlPID

# Show all properties of a unit
systemctl show nginx
systemctl show nginx --property=ExecStart --property=Restart

# Check unit file loading order and dependencies
systemd-analyze dump | grep -A 5 "nginx.service"

# View the effective (merged) configuration
systemctl cat nginx

# Check for override/drop-in files
systemd show nginx -p DropInPaths

# View cgroup resource usage for a service
systemd-cgtop

# Show resource usage for a specific service slice
systemctl show myapp.service -p MemoryCurrent -p CPUUsageNSec

# Reset a failed state (must do this before restarting)
sudo systemctl reset-failed

# List all currently failed units
systemctl --failed
```

Service Hardening Reference

The most impactful hardening options ranked by security value:

```
[Service]
# === Essential (apply to every service) ===
NoNewPrivileges=yes
ProtectSystem=strict
PrivateTmp=yes

# === Strongly recommended ===
ProtectHome=true
PrivateDevices=yes
ProtectKernelTunables=yes
ProtectControlGroups=yes
ReadWritePaths=/var/lib/myapp     # only if the service needs write access
```

```
# === High security ===
ProtectKernelModules=yes
SystemCallFilter=@system-service
SystemCallErrorNumber=EPERM       # deny blocked syscalls instead of killing
MemoryDenyWriteExecute=yes
CapabilityBoundingSet=CAP_NET_BIND_SERVICE  # only grant what's needed

# === Maximum isolation ===
PrivateNetwork=yes               # no network at all (good for cron-like tasks)
PrivateUsers=yes                 # separate UID namespace
LockPersonality=yes
RemoveIPC=yes
RestrictAddressFamilies=AF_INET AF_INET6  # restrict socket families
RestrictRealtime=yes
DynamicUser=yes                  # allocate a temporary user (no static User= needed)
```

Environment Files

```
# /etc/myapp/environment — systemd environment file format
# Lines are KEY=VALUE pairs
# Empty lines and lines starting with # or ; are ignored
# No quoting needed — values are taken literally (including spaces)

NODE_ENV=production
DATABASE_URL=postgres://user:pass@localhost:5432/mydb
LOG_LEVEL=info
CACHE_SIZE=1024
```

```
# In the service unit, reference the file:
EnvironmentFile=/etc/myapp/environment

# Multiple files — later files override earlier ones:
EnvironmentFile=/etc/myapp/environment
EnvironmentFile=/etc/myapp/environment.local

# Dash prefix means "don't fail if missing" (for optional local overrides):
EnvironmentFile=-/etc/myapp/environment.local

# Individual Environment= directives take precedence over EnvironmentFile
Environment="NODE_ENV=staging"  # this overrides NODE_ENV from the file
```

ExecStart Pre/Post Scripts

```
[Service]
Type=oneshot
RemainAfterExit=yes

# Run before the main command — all must succeed
ExecStartPre=/bin/mkdir -p /var/lib/myapp
ExecStartPre=/usr/bin/chown -R myapp:myapp /var/lib/myapp
ExecStartPre=/usr/bin/myapp --validate-config

# The main command
ExecStart=/usr/bin/myapp --init
```

```
# Run after the main command starts (runs in parallel with the main process)
ExecStartPost=/usr/bin/myapp --register-health-check

# Run when the service is stopped
ExecStop=/usr/bin/myapp --graceful-shutdown

# Run after the service stops (cleanup)
ExecStopPost=/usr/bin/rm -f /run/myapp.pid
```

```
# Prefix a command with - (dash) to make failures non-fatal
ExecStartPre=-/usr/bin/touch /var/log/myapp/boot.log
```

Resource Limits Reference

```
[Service]
# CPU
CPUQuota=200%                  # 2 CPU cores max
CPUQuota=50%                   # half a core
CPUWeight=100                  # relative CPU share (default 100)
CPUStartupWeight=200           # higher share during startup

# Memory
MemoryMax=1G                   # hard limit — OOM if exceeded
MemoryHigh=768M                # soft limit — throttled but not killed
MemoryMin=64M                  # guaranteed minimum
MemorySwapMax=512M             # swap limit
MemoryZSwapMax=256M            # compressed swap limit

# File descriptors
LimitNOFILE=65536              # open files (soft = hard = 65536)
# Fine-grained: LimitNOFILE=1024:65536  (soft:hard)

# Processes and threads
LimitNPROC=4096                # max processes per user
TasksMax=512                   # max threads in the cgroup

# File size
LimitFSIZE=100M                # max file size the process can create

# Locks
LimitLOCKS=infinity            # max file locks

# Stack size
LimitSTACK=8M                  # max stack size per thread

# Wall clock time
LimitCPU=2h                    # max CPU time (process killed if exceeded)

# Number of file locks
LimitLOCK=1024
```

Quick Reference Card

```
# === Service lifecycle ===
systemctl start|stop|restart|reload SERVICE
systemctl enable|disable|is-enabled SERVICE
systemctl enable --now SERVICE       # enable + start
systemctl disable --now SERVICE      # disable + stop
systemctl status SERVICE
systemctl mask|unmask SERVICE

# === Listing ===
systemctl list-units --type=service
systemctl list-unit-files --type=service
systemctl list-timers --all
systemctl list-dependencies SERVICE
systemctl --failed

# === Editing ===
systemctl edit SERVICE              # create drop-in override
systemctl cat SERVICE               # show resolved unit file
systemctl daemon-reload             # reload after editing unit files

# === Targets (runlevels) ===
systemctl get-default
systemctl set-default TARGET
systemctl isolate TARGET

# === journalctl ===
journalctl -u SERVICE               # logs for a service
journalctl -b                       # current boot
journalctl -b -1                    # previous boot
journalctl --since "1 hour ago"
journalctl -p err                   # errors and above
journalctl -f                       # follow
journalctl -g "pattern"             # grep
journalctl --disk-usage
journalctl --vacuum-size=500M
journalctl --vacuum-time=2weeks

# === Troubleshooting ===
systemd-analyze verify UNIT_FILE
systemd-analyze blame
systemd-analyze critical-chain SERVICE
systemctl reset-failed
```