



# Das Init-System Systemd, Teil 1

 Thorsten Leemhuis , Kay Sievers, Lennart Poettering

## Linux, Systemd

09.11.2020, 16:09

*Artikelserie beschreibt den praktischen Umgang mit Systemd.*

### **Zu den Autoren**

Lennart Poettering und Kay Sievers haben Systemd entworfen und entwickelt. Während der Entstehung des Artikels arbeitete Poettering bei Red Hat und Sievers bei Novell.

Bei Linux-Distributionen übergibt der Kernel traditionell [Sysvinit](#) die Verantwortung zur Einrichtung des Systems. Einige Jahre sah alles danach aus, als würde [Upstart](#) das angestaubte, aber noch weitverbreitete Init-System beerben, doch mittlerweile immer mehr Distributionen auf [Systemd](#) um – [abgesehen von Ubuntu](#), das laut Mark Suttleworth auf absehbare Zeit bei Upstart bleiben wird. Fedora nutzt Systemd seit Version 15, auch in OpenSuse 12.1 und Mandriva 2011 kommt das neue Init-System zum Einsatz; Mageia steigt mit Version 2 um. Bei Arch Linux und Gentoo sowie Debian Testing liegt Systemd bei; bei einigen weitere Distributionen wird der optionale Einsatz oder der Umstieg diskutiert.

Eine der Besonderheiten von Systemd ist der parallele Start von Hintergrunddiensten, ohne dass Abhängigkeiten zwischen diesen explizit festgelegt werden müssen; das nutzt Hardware-Ressourcen effizienter und lässt das System flott starten. Systemd erledigt zudem einige Aufgaben, um die sich bislang meist distributionsspezifische Skripte kümmern; ganz nebenbei beseitigt es damit einige Unterschiede bei der Bedienung und Konfiguration von Distributionen.

## **Aufgaben**

Der Init-Prozess ist der erste Prozess, den der Kernel erzeugt. Alle weiteren Prozesse sind Kinder des Init-Prozesses, der daher die Verantwortung für die komplette Einrichtung des Userlands trägt. Dazu gehört nicht nur das Einhängen von Dateisystemen und die Netzwerkeinrichtung, sondern auch das Starten von Hintergrund-Diensten und Programmen – darunter auch jene, über die sich Benutzer am System anmelden.

Nach dem Abschluss der Systemeinrichtung läuft der Init-Prozess weitgehend untätig im Hintergrund weiter. Er kommuniziert mit dem Kernel und wird beispielsweise informiert, wenn der Benutzer Strg+Alt+Entf drückt. Genau wie beim Aufruf von Befehlen wie `shutdown -r now` oder `reboot` erledigt der Prozess mit der PID 1 dann alles Nötige, um das System sauber zum Stillstand zu bringen.

Mit diesen Aufgaben wurde in den 80er Jahren in Unix System V das einfache, aber flexible "System V Init System" betraut. In den 90er Jahren entstand eine Sysvinit genannte Neuimplementierung dieses Init-Systems. Sie arbeitet mit einer ganz ähnlichen Logik und kommt bis heute bei vielen Linux-Distributionen zum Einsatz. Sysvinit erledigt die Aufgaben des Systemstarts im Wesentlichen mit Shell-Skripten, die einfach der Reihe nach abgearbeitet werden.

Mit der Verbreitung von Linux in Mobilgeräten, Desktop-PCs, Fernsehern und zahlreichen

anderen Gebieten wandelten sich allerdings die Anforderungen an den Init-Prozess: Der Systemstart sollte flexibler werden und dank Parallelisierung deutlich schneller ablaufen.

## Herangehensweisen

Lange schien es, als wäre [das 2006](#) von einem Canonical-Entwickler gestartete Upstart der designierte Nachfolger für Sysvinit (siehe den Artikel [Schneller booten mit Upstart](#) auf [heise open](#)). Anfangs kam es nur bei Ubuntu zum Einsatz, später auch bei Fedora (Versionen 9 bis 14) und Red Hat Enterprise Linux (RHEL6-Familie). OpenSuse experimentierte während der Arbeit an der Version 11.3 mit Upstart, blieb letztlich jedoch bei Sysvinit.

Upstart ist ein ereignisorientiertes Init-System – es kann Dienste starten, wenn Ereignisse wie "Netzwerk ist konfiguriert" oder "Netzlaufwerk ist eingebunden" eintreten. Der Ansatz unterscheidet sich stark von dem statischen Sysvinit, daher lassen sich bestehende Konfigurationen nur schwer oder gar nicht in das Ereignis-Modell von Upstart übertragen.

Im [April 2010](#) erschien Systemd; es bedient sich einiger Ideen aus früheren Unit-Systemen und kombiniert diese mit einer einheitlichen Konfigurations- und Administrationsschnittstelle. Systemd arbeitet als Hintergrunddienst (Daemon) und steuert wichtige Aspekte der Systemkonfiguration von der Initialisierung der Hardware bis zu den gestarteten Server-Prozessen. Der Name erschien den Entwicklern als eine [passende Verbindung](#) zum französischen Begriff "Système D" (etwa: "Trick 17"), der kreative technische Lösungsansätze à la [MacGyver](#) beschreibt.

## Socket-Aktivierung

### Hintergrund

Zentrales Merkmal von Systemd ist die Socket-Aktivierung, durch die der Daemon Hintergrunddienste ohne explizite Konfiguration der Abhängigkeiten parallel starten kann, sobald die Grundeinrichtung des Systems abgeschlossen und die lokalen Dateisysteme eingebunden sind. Der Trick besteht darin, dass Systemd die Sockets zur Kommunikation mit den zu startenden Diensten selbst anlegt und dorthin geschriebene Daten zwischenspeichert, bis sie der gestartete Dienst entgegennehmen kann.

Illustrieren lässt sich das Konzept am Beispiel der Dienste Syslog und D-Bus. Letzterer verbindet sich beim Start mit dem Socket `/dev/log`, um darüber bei Bedarf Status- und Fehlermeldungen über den Log-Daemon ins Systemlog zu schreiben. Sysvinit kann daher D-Bus erst starten, wenn der Syslog-Dienst voll einsatzbereit ist.

Systemd hingegen legt `/dev/log` selbst an und startet Syslog und D-Bus gleichzeitig; dabei werden die Daten, die D-Bus an `/dev/log` schickt, gepuffert, bis sie Syslog entgegennimmt. Anfangs überließ Systemd dem Kernel das Puffern; bei aktuellen Versionen vom Systemd kümmert sich die [dort enthaltene](#) Log-Funktion "Journal" um diese Aufgabe.

Systemd kann so auch Bluetooth, Avahi und weitere Dienste, die mit dem Log-Daemon oder D-Bus kommunizieren, parallel mit Syslog und D-Bus starten. Wenn Avahi beim Start eine Antwort von D-Bus erwartet, stoppt der Prozess an dieser Stelle automatisch und setzt den Start ohne weiteres Zutun fort, sobald die Antwort über den Socket eintrifft. Sollte D-Bus aus irgendeinem Grund nicht anlaufen, bricht Systemd den Start von Bluetooth und Avahi nach einiger Zeit ab.

```
Started Job spooling tools.
Starting SYSU: Init script for live image...
Starting System Logging Service...
Starting ABRT Automated Bug Reporting Tool...
Starting Avahi mDNS/DNS-SD Stack...
Starting irqbalance daemon...
Starting Command Scheduler...
Started Command Scheduler.
Starting Machine Check Exception Logging Daemon...
Starting System Setup Keyboard...
Started System Setup Keyboard.
Starting D-Bus System Message Bus...
Started Restore Sound Card State.
Started Console System Startup Logging.
Started Bootup unback.
Started irqbalance daemon.
Stopping Syslog Kernel Log Buffer Bridge...
Started D-Bus System Message Bus.
Stopped Syslog Kernel Log Buffer Bridge.
Started System Logging Service.
Started Machine Check Exception Logging Daemon.
Started Avahi mDNS/DNS-SD Stack.
Started ABRT Automated Bug Reporting Tool.
Started Network Manager.
```

Durch Socket-Aktivierung ändert sich gelegentlich die Reihenfolge, in der die Dienste den Start bestätigen. Hier vermeldet der Irqbalance-Daemon etwa vor D-Bus und Syslogd den Start; beim nächsten Boot war es genau andersherum.

Apple verwendet dasselbe Prinzip in seinem mit Mac OS X 10.4 eingeführten [Launchd](#), der auch bei iOS zum Einsatz kommt und als Hauptgrund für den deutlich verkürzten Startprozess neuerer Mac-OS-Versionen gilt, da der Parallelstart der Dienste die verfügbaren CPU- und I/O-Ressourcen effizienter nutzt. Bei Sysvinit starten die Dienste hingegen in einer festgelegten Reihenfolge – Bluetooth und Avahi erst, wenn der D-Bus-Daemon läuft, der wieder erst startet, wenn das Syslog bereit ist. Selbst Bluetooth und Avahi, die voneinander unabhängig sind, starten nicht bei allen Sysvinit-Distributionen parallel.

## Bedarfsanforderung

Da Systemd den Socket erstellt und hält, kann der Daemon einen abgestürzten Dienst neu starten, ohne dass mit dem Socket verbundene Programme die Verbindung verlieren. Dadurch lassen sich System-Komponenten einfacher aktualisieren, da der Kernel die über den Socket eingehenden Client-Anfragen während des Neustarts puffert und der neue Dienst einfach dort fortfahren kann, wo der alte aufgehört hat.

Sockets lassen sich zudem an verschiedene Programme übergeben. Systemd nutzt das

zum Loggen von früher Statusmeldungen: Solange das Root-Dateisystem noch nicht beschreibbar eingebunden ist, nimmt ein minimaler Log-Dienst Daten entgegen, die nach `/dev/log` geschrieben werden, und schreibt sie in den Kernel-Log-Puffer. Sobald der eigentliche Syslog-Server bereits ist, wird der Minidienst beendet; der Syslog-Daemon übernimmt den Socket und schreibt dabei alle zuvor aufgelaufenen Nachrichten aus dem Kernel-Log-Buffer auf die Festplatte. So gehen keine Nachrichten verloren, was eine Protokollierung vom ersten Moment des Bootens an ermöglicht.

## Units und Targets

### Einheiten

Die verschiedenen Tätigkeiten beim Systemstart – Sockets anlegen, Hardware einrichten, Datenträger einbinden, Hintergrunddienste starten und so weiter – sind in sogenannten Units organisiert. Für jede Aufgabe, die Systemd ausführen soll, benötigt man eine Unit-spezifische Konfigurationsdatei – bei einer Mount-Unit beispielsweise muss die Konfiguration lediglich die Device-Datei des Datenträgers und das Zielverzeichnis enthalten. Diese Unit-Dateien sind erheblich kürzer als traditionelle Init-Skripte. Syntaktisch ähneln sie den Ini-Dateien von Windows.

Den Typ einer Unit erkennt Systemd am Dateinamen. Dateien, die auf `.service` enden, legen Service-Units an; sie kümmern sich um Hintergrunddienste. Units zum Ein- und Aushängen von Dateisystemen enden auf `.mount`; das Suffix lautet `.automount`, wenn dabei der Automounter involviert ist, der Dateisysteme beim Zugriff automatisch einhängt. Units mit dem Suffix `.path` weisen Systemd an, die spezifizierten Dateien und Verzeichnisse via Inotify überwachen; erfolgt dort ein Zugriff, startet Systemd diese Unit.

Auf `.socket` endende Unit-Dateien legen einen oder mehrere Sockets für die Socket-Aktivierung an. Der zugehörige Dienst wird erst dann über eine zu der Socket-Unit gehörende Service-Unit gestartet, wenn ein anderer Prozess Daten auf den Socket schreibt. Der geöffnete Socket wird dabei an den Dienst übergeben – ähnlich wie es alte Unix-/Linux-Hasen von Inetd kennen.

```

cttest@localhost ~$ systemctl --type=service | head -n 20 ; echo ; systemctl --type=service | tail -n 20
abrt.ccpp.service loaded active exited LSB: installs coredump handler which saves a
default data
abrt.service loaded active running ABRT Automated Bug Reporting Tool
accounts-daemon.service loaded active running Accounts Service
acpid.service loaded active running ACPI Event Daemon
atd.service loaded active running Job spooling tools
auditd.service loaded active running SYSV: This starts the Linux Auditing System
Daemon, which collects security related events in a dedicated audit log. If this daemon is turne
d off, audit events will be sent to syslog.
console-...daemon.service loaded active running Console Manager
console-...start.service loaded active exited Console System Startup Logging
cpufreq.service loaded active exited LSB: processor frequency scaling support
crond.service loaded active running Command Scheduler

systemd-...collect.service loaded active exited Collect Read-Ahead Data
systemd-...replay.service loaded active exited Replay Read-Ahead Data
systemd-...pi-wfs.service loaded active exited Reaccount AP[ wfs
systemd-...synctl.service loaded active exited Apply Kernel Variables
systemd-...-setup.service loaded active exited Recreate Volatile Files and Directories
systemd-...actions.service loaded active exited Permit User Sessions
systemd-...-setup.service loaded active exited Setup Virtual Console
udev-settle.service loaded active exited udev Wait for Complete Device Initialization
udev-trigger.service loaded active exited udev Coldplug all Devices
udev.service loaded active running udev Kernel Device Manager
cttest@localhost ~$

```

Service-Units kümmern sich um Hintergrunddienste.

Die Unit-Dateien von Systemd und den Systemdiensten liegen im Verzeichnis `/lib/systemd/system/`; liegt eine gleichnamige Datei in `/etc/systemd/system/`, ignoriert Systemd die im Lib-Verzeichnis. Der Administrator kann so eine Unit-Datei von Systemd kopieren und an seine Belange anpassen, ohne fürchten zu müssen, dass sie beim nächsten Update überschrieben wird – das konnte bei Sysvinit-Distributionen passieren, wenn man eines der in `/etc/rc.d/init.d/` gespeicherten Init-Skripte von Hand veränderte.

Systemd erzeugt einige Units dynamisch selbst; sie tauchen daher nicht im Dateisystem, sondern nur in der mit `systemctl` abrufbaren Liste der Units auf. So wird für einige Geräte wie Datenträger, PCI-Geräte und TTYs im Zusammenspiel mit Udev automatisch eine Device-Unit generiert, wenn sie in den Udev-Regeln mit `TAG+="systemd"` gekennzeichnet sind. Ähnlich wie beim Zweigespann aus Socket- und Service-Unit können andere Units von diesen Device-Units abhängen und so automatisch starten, wenn Geräte auftauchen, auf die sie angewiesen sind.

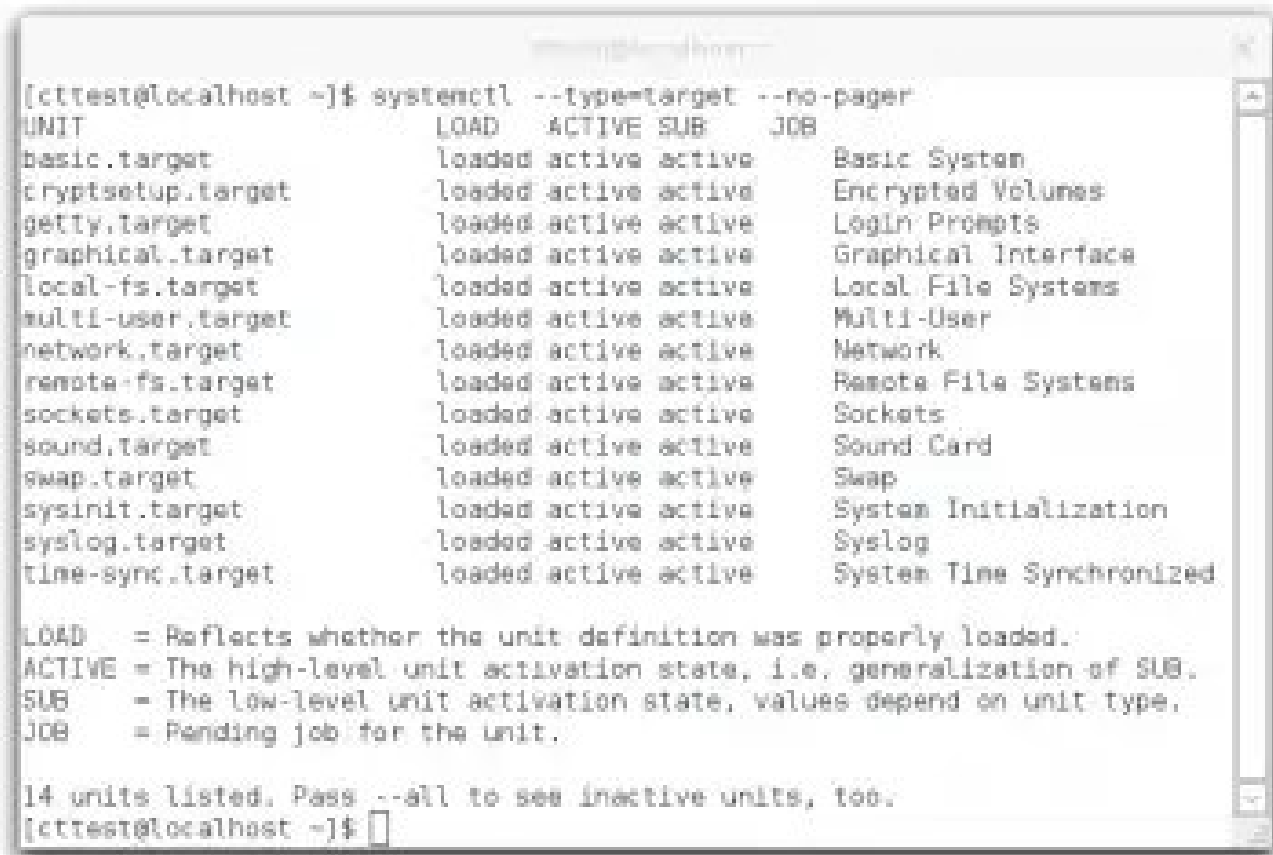
Dieses System kommt auch bei den `.swap`-Units zum Einsatz, die automatisch mit Hilfe der Angaben in `/etc/fstab` angelegt werden und die den Auslagerungsspeicher einbinden, sobald das spezifizierte Swap-Volume auftaucht. Systemd erzeugt auch für einige andere in `/etc/fstab` spezifizierte Datenträger automatisch Mount-Units, daher tauchen in der `Systemctl`-Liste mehr Mount-Units auf, als man Konfigurationsdateien findet.

## Ziele

Unit-Dateien, die auf `.target` enden, definieren Gruppen aus Units. Sie leisten selbst wenig, rufen vielmehr andere Units auf, die für Dienste, Dateisysteme und andere Dinge zuständig sind. So lassen sich Boot-Ziele definieren, die den klassischen Sys-V-Runlevels



entsprechen. Die Unit `multi-user.target` etwa sorgt für den Start all jener Dienste, die ältere Fedora- und OpenSuse-Versionen im Runlevel 3 aufrufen würden – voller Multiuser-Netzwerkbetrieb ohne grafischen Anmeldemanager. Letzterer erscheint bei der Unit `graphical.target`, die damit das Äquivalent zum Runlevel 5 darstellt und typischerweise das Standard-Ziel ist.



```
[cttest@localhost ~]$ systemctl --type=target --no-pager
UNIT                                LOAD    ACTIVE SUB    JOB
basic.target                       loaded active active   Basic System
cryptsetup.target                  loaded active active   Encrypted Volumes
getty.target                        loaded active active   Login Prompts
graphical.target                   loaded active active   Graphical Interface
local-fs.target                    loaded active active   Local File Systems
multi-user.target                  loaded active active   Multi-User
network.target                     loaded active active   Network
remote-fs.target                   loaded active active   Remote File Systems
sockets.target                     loaded active active   Sockets
sound.target                       loaded active active   Sound Card
swap.target                        loaded active active   Swap
sysinit.target                     loaded active active   System Initialization
syslog.target                      loaded active active   Syslog
time-sync.target                   loaded active active   System Time Synchronized

LOAD    = Reflects whether the unit definition was properly loaded.
ACTIVE  = The high-level unit activation state, i.e. generalization of SUB.
SUB     = The low-level unit activation state, values depend on unit type.
JOB     = Pending job for the unit.

14 units listed. Pass --all to see inactive units, too.
[cttest@localhost ~]$
```

Target-Units starten über Abhängigkeiten andere Units und sind mit klassischen Sys-V-Runleveln vergleichbar.

Beim Hochfahren des Systems aktiviert Systemd die spezielle Target-Unit `default.target`, typischerweise ein Alias eines anderen Targets wie `graphical.target` oder `multi-user.target`. Die Targets können zudem aufeinander aufbauen oder voneinander abhängen; `graphical.target` etwa wartet den Start von `multi-user.target` ab, bevor es die grafische Oberfläche startet.

Wo nötig lassen sich über die Angabe von "Wants" in den Unit-Dateien manuell Abhängigkeiten zwischen den Units definieren – wichtig beispielsweise für Dienste wie den Apache-Webserver, der beim Start eine voll konfigurierte Netzwerkumgebung erwartet. Solche Dienste sollten von `network.target` abhängen. Bei Diensten wie Avahi oder Bind ist eine solche Abhängigkeit nicht nötig, da diese auch mit Netzwerk-Interfaces zurechtkommen, die zur Laufzeit erscheinen oder verschwinden.

## SysV-Kompatibilität, Control Groups

### Althergebrachtes

Aus Kompatibilitätsgründen versteht sich Systemd auch mit System-V- und LSB-Init-Skripten, die nicht nur von Sysvinit-Distributionen verwendet werden, sondern auch mit Upstart funktionieren. Diese Init-Skripte werden durch eine Shell interpretiert und erfordern einen Parameter wie "start", "stop" oder "restart".

Auch die Hersteller kommerzieller Software legen ihren Hintergrunddiensten typischerweise Sys-V- und LSB-Init-Skripte bei. Um sie intern wie eine richtige Service-Unit zu behandeln, generiert Systemd daraus automatisch eine Service-Unit; das Init-Tool ignoriert allerdings Sys-V- und LSB-Init-Skripte, wenn es eine Unit-Datei mit gleichem Namen findet.

## Gruppen

Systemd packt jeden Dienst beim Start in eine eigene, nach dem Dienst benannte **Control Group**. Diese Technik isoliert die Prozesse und bietet mit Hilfe **optionaler Controller** Stellschrauben, um die Verteilung der Ressourcen zu beeinflussen. Kindprozesse erben die Gruppenzugehörigkeit.

Systemd kann so Prozessgruppen als zusammengehörige Einheiten verwalten, um etwa beim Beenden eines Dienstes alle zugehörigen Prozesse zuverlässig zu stoppen. Administratoren können über die normalen Cgroup-Schnittstellen den Ressourcenverbrauch von Diensten kontrollieren; die manuelle Zuordnung der Prozesse entfällt.

## Breiterer Ansatz

Systemd liegen eine Reihe von Units bei, die einige grundsätzliche Dinge bei der Initialisierung des Systems erledigen. Teilweise sind diese wie ein Hintergrunddienst angelegt. Die Service-Unit `fsck-root.service` etwa veranlasst bei Bedarf eine Prüfung des Root-Dateisystems, bevor es durch `remount-rootfs.service` beschreibbar eingebunden wird. Die Service-Units `hwclock-load` und `hwclock-save` sorgen für einen Abgleich der Zeit mit der Systemuhr.

Bei Sysvinit- und Upstart-Distributionen kümmern sich Shell-Skripte um derartige Aufgaben, etwa `/etc/rc.sysinit` oder eine Sammlung kleiner Skripte in `/etc/rcS.d/`. Diese Skripte sind stark auf die jeweilige Distributionsfamilie zugeschnitten und verhalten sich daher bei Debian ganz anders als bei Fedora oder OpenSuse; das ist der Grund, warum man bei Fedora und RHEL in `/etc/sysconfig/keyboard` die Tastatur festlegen kann, dieses Verzeichnis bei Debian aber vergeblich sucht.

Viele Systemd-Units starten C-Programme, die schneller und robuster sein sollen als die Shell-Skripte, die diese Aufgaben bisher erledigten. Mit der Integration dieser Dienste versucht Systemd viele Unterschiede zwischen den Distributionen aus der Welt zu schaffen. Das erleichtert Entwicklern die Arbeit, denn sie können Unit-Dateien für ihre Dienste mitliefern und dabei die Dinge erwarten, die Systemd beiliegt. Sys-V-Init-Skripte beizulegen ist erheblich schwieriger, weil diese auf distributionsspezifische Eigenarten



Rücksicht nehmen müssen.

## Details

Weitere Hintergründe zu Ideen, Arbeitsweisen und Einsatz von Systemd liefern die [Man-Pages zu Systemd](#) – etwa [systemd](#) und [systemd.conf](#). Auch für jeden der Unit-Typen gibt es Man-Pages – beispielsweise [systemd.unit](#) oder [systemd.service](#). Über die [Homepage von Lennart Poettering](#) findet man [zahlreiche Artikel](#), die Hintergründe zum Init-System erläutern, darunter die bislang zwei Teile umfassende Serie "systemd for Developers":

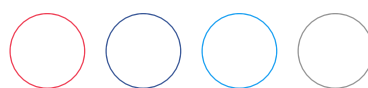
- [Part I: Socket Activation](#)
- [Part II: Socket Activation \(Part II\)](#)

Im [dritten "Systemd Status Update"](#) hat Poettering zudem vor Kurzem einige der Neuerungen aufgelistet, die in den letzten eineinhalb Jahren in Systemd eingeflossen sind.

Im [zweiten Teil des Artikels](#) geht es um Systemd aus Adminsicht: Aufbau von Unit-Dateien, Befehle zum Auflisten, Starten und Stoppen von Units, Statusausgaben und Beheben von Problemen. ([thl](#))

Forum zum Thema: [Betriebssysteme](#)

TEILE DIESEN BEITRAG



<https://heise.de/-1563259>

[Drucken](#)

## Auch interessant



Anzeige

**Solaranlage + Speicher: Staat gibt unglaublichen Anreiz!**

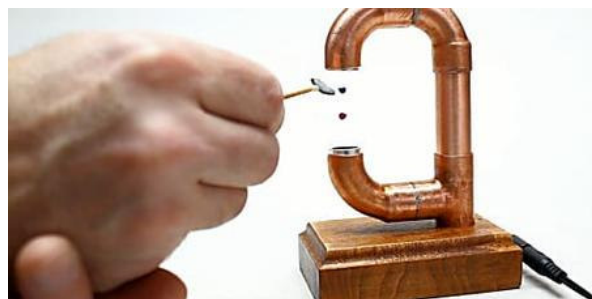
Solaranlagen



Anzeige

**Auto Abo: Nur noch Tanken – alles andere ist inklusive.**

Care by Volvo



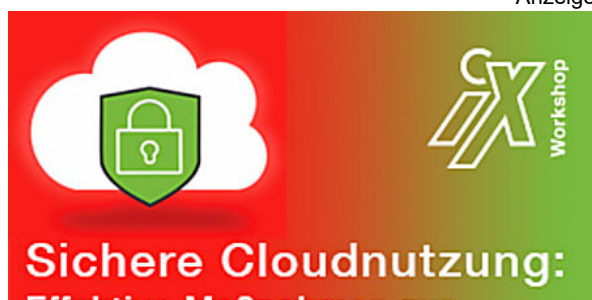
## Einfacher Ultraschall-Levitationsapparat

Make:



## Ihre zertifizierte Weiterbildung durch hochwertige Online-Trainings mit hohem Praxisbezug.

heise.de



## Der Kurs vermittelt an zwei Tagen das nötige Know-how, um Cloud-Dienste sicher und effizient im...

heise-events.de



## Agentur entschuldigt sich für VW-Werbepot

heise online

empfohlen von

## KOMMENTARE

[Kommentare lesen \(209 Beiträge\)](#)

Stöbern

Suchbegriff



Test & Kaufberatung

Praxis & Tipps

Wissen

Trends & News

@ctmagazin



c't daily

Kontakt

c't-Projekte

Service

Archiv vor 2012

Werben auf c't

Impressum

Newsletter

Leserforum allgemein

Blog

Jahresarchiv

RSS-Feed

Datenschutz



---

Datenschutzhinweis

Impressum

Kontakt

Mediadaten

832500

Content Management by **InterRed**

Copyright © 2020 Heise Medien