3.4 Grundkonfiguration/Systemstart

3.4.1 Systemstart bei Linux

In Unix-artigen System hängen alle Prozesse baumartig zusammen. Jeder Prozess kann neue Prozesse hervorrufen. Der einzige Prozess, der *keine* Eltern hat, ist der `init`-Prozess (nähere Infos erhält man mit `man 8 init`). Er ist der erste Prozess und hat daher die Prozess-ID #1. Er wird beim Systemstart „von Hand“ in die Prozesstabelle eingetragen[1].

Woher bekommt der `init`-Prozess nun seine Daten, wie er das System starten soll, also welche Programme er in welcher zeitlichen Reihenfolge aufruft?

Wie so oft im Linux-Umfeld hat man da mehrere Möglichkeiten.

- Die System-V-Init-Methode (kurz: sysvinit) hat sich seit vielen Jahren bewährt. Sie basiert auf einer Konfigurationsdatei (`/etc/inittab`) und vielen Skripten und Verknüpfungen. Für Server-Systeme ist sie sehr gut geeignet, da sie dem Administrator viele Freiheiten lässt. sysvinit kann ohne große Änderungen auf nahezu jedem Unix-artigen System eingesetzt werden.
- Die Systemd-Methode ist relativ neu und auf Linux zugeschnitten. Sie erlaubt es, dass schon beim Start mehrere Prozesse gleichzeitig hochgefahren werden; es wird erreicht, dass jeder Prozess erst dann gestartet wird, wenn alle Prozesse, von denen er abhängt, ihre Aufgabe erfüllen. Dadurch kann ein Rechner mit der systemd-Methode etwas schneller starten. Somit ist systemd für Desktop-, Notebook- und Tablet-PCs geeignet. In der Kritik steht systemd, weil es Aufgaben übernimmt, die eigentlich nicht für einen init-Prozess gedacht sind. Bei Debian ist Systemd seit Version 8 Standard[2].
- Außerdem gibt es noch die Methoden Upstart (ehemals Ubuntu und Fedora), launchd (MacOS X), OpenRC (gentoo), initNG und cinit.

3.4.2 Sys-V-Init

3.4.2.1 Sys-V-Init, Teil 1: inittab Bei der Methode nach Unix-System V findet alle Systemkonfiguration zunächst über die Datei `/etc/inittab` statt (Abbildung 1). Hier liest `init`

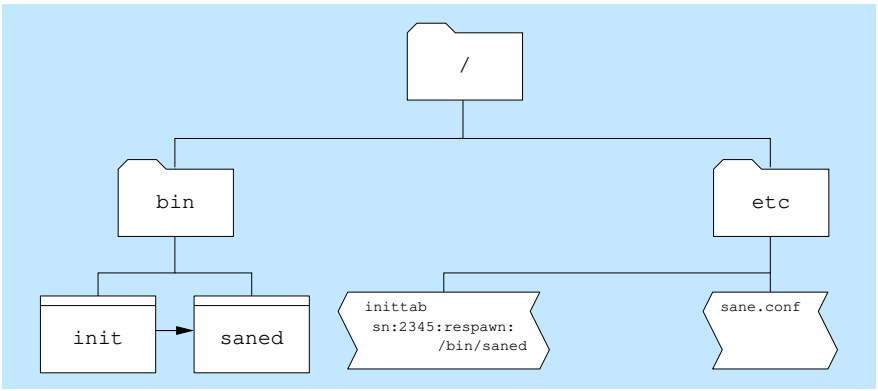

Abbildung 1: Sys-V-Init und inittab

aus seiner Konfigurationsdatei `/etc/inittab` von oben nach unten alles heraus, was es tun soll. Jede Zeile entspricht einem Eintrag[3].

Jeder Eintrag in der `inittab` hat die Form

[1] Man kann dem Bootmanager jedoch mitteilen, dass er ein anderes Programm starten soll.

[2] Die Distribution Devuan bietet ein Debian-ähnliches Linux ohne Systemd.

[3] Beim Ansehen von `/etc/inittab` sieht man, dass viele Einträge enthalten den Programmnamen `getty` enthalten. Programme dieser Art starten in der Regel Login-Möglichkeiten für Kanäle, auf denen sich Benutzer einloggen sollen (Konsole und serielle Leitungen). TCP/IP-Kanäle werden durch andere Dienste verwaltet.

```
1 ID:Runlevels:Aktion:Prozess
```

Eine der `inittab`-Zeilen in einem unmodifizierten Debian-System hat beispielsweise den Inhalt:

```
1 sn:2345:respawn:/bin/saned
```

Hier die Bedeutung der einzelnen Felder:

- `sn`: ID – eindeutige Kennzeichnung, maximal 2 Zeichen (oder 4, je nach init-Version). Eine Besonderheit: Bei `getty`-Zeilen muss die `id` aus Kompatibilitätsgründen gleich der Bezeichnung der entsprechenden seriellen Schnittstelle bzw. virtuellen Konsole sein (die serielle Schnittstelle mit der Gerätedatei `/dev/ttyS0` hat die id "`S0`"; virtuelle Konsolen mit den Gerätedateien /dev/tty1,2, .. haben die id "1", "2", ..).
- `2345`: Runlevels – Software-Konfigurationen, die bestimmte Arten von Prozessen erlauben. Traditionell sind die in Tabelle 1 aufgeführten Konfigurationen vorgesehen. Eine Zeile wird

Runlevel	Bedeutung
0	System-Halt
1, S	Single-User-Modus, für Wartungsarbeiten
2	Multi-User-Modus ohne Netzwerk
3	Multi-User-Modus mit Netzwerk
4	frei
5	Multi-User-Modus mit Netzwerk und grafischem Login
6	System-Reboot
7, 8, 9	frei

Tabelle 1: Runlevel

  nur dann ausgeführt, wenn der entsprechende Runlevel im runlevel-Feld dieser Zeile enthalten ist.

- `respawn`: Aktion – In diesem Feld ist enthalten, was mit dem Prozess passieren soll:
  - boot - unabhängig vom runlevel wird der Prozess beim Booten gestartet.
  - bootwait - unabhängig vom runlevel wird der Prozess beim Booten gestartet; init wartet auf das Ende des Prozesses.
  - wait - das Programm wird einmal gestartet und init wartet auf das Ende des Prozesses.
  - once - das Programm wird einmal gestartet. Init macht unterdessen weiter.
  - respawn -sobald das Programm terminiert hat, wird es wieder angeworfen.
  - off - nichts passiert.
  - initdefault - gibt nur an, welcher runlevel nach dem Booten gestartet werden soll.
- `/bin/saned`: Prozess – Hier steht eine Befehlszeile, eventuell noch mit Parametern. Der Programmname muss mit vollständigem absolutem Pfadnamen eingetragen sein, denn `init` kennt möglicherweise nicht die Shell-Variable `$PATH` (genauso ist es mit `$COLS`, `$LINES`). Das aktuelle Verzeichnis ist auch nicht bekannt. Außerdem ist daran zu denken, dass nicht `bash`, sondern die eher spartanische Shell `sh` diese Befehlszeile interpretiert.

Auf diese Weise kann man in `inittab` auch selbst weitere Zeilen eintragen.

Den aktuellen Runlevel kann man einfach erfragen:

Terminal

```
root@debian964:~# runlevel
S 2
```

Der aktuelle Runlevel ist hier 2, der vorherige war S.

Mit dem folgenden Befehl kann man als Administrator das System in einen anderen Runlevel bringen[4]:

Terminal
```
root@debian964:~# telinit 3 # Wechsel von Runlevel 2 nach 3
```

Falls man als neuen Runlevel den aktuellen Runlevel angibt (Wechsel von 2 nach 2), dann passiert gar nichts (es wird nicht einmal eine Warnmeldung ausgegeben).

Wenn man also die `inittab` für den aktuellen Runlevel neu einlesen will, muss das anders gemacht werden, und zwar so:

Terminal
```
root@debian964:~# telinit q # Aktuellen Runlevel neu einlesen
```

3.4.2.2 Sys-V-Init, Teil 2: rc.d-Startskripten Mit der `inittab`-Mechanik kann man schon einmal Prozesse starten lassen.

Richtig flexibel wird System-V-Init aber erst durch die so genannten Startskripte, die es erlauben, beim Umschalten zwischen verschiedenen Runleveln gezielt und in der richtigen Reihenfolge bestimmte Server zu stoppen und andere zu starten (Abbildung 2). Wieder beginnt alles mit dem

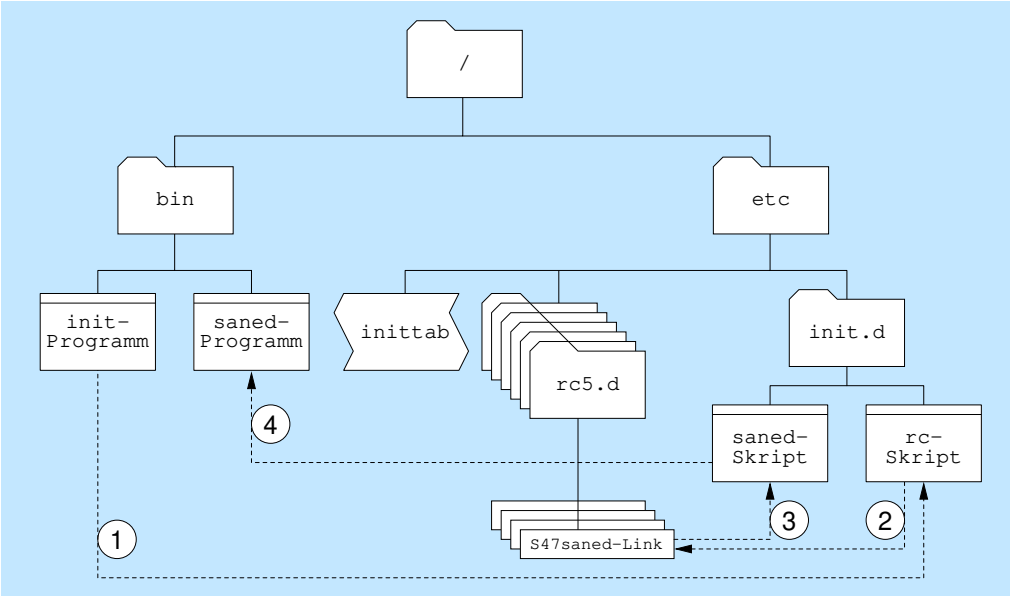

Abbildung 2: System-V-Init und Startskripte

`init`-Prozess; noch ziemlich am Beginn der Datei steht die Zeile:

```
1 l5:5:wait:/etc/init.d/rc 5
```

Falls also der Runlevel 5 gewünscht wurde, wird nun das Programm `rc` mit dem Parameter 5 aufgerufen.

`rc` ist ein Shellscript, welches in das Verzeichnis `/etc/init.d/rc${1}.d` wechselt und in nach ASCII sortierter Reihenfolge alle dort vorhandenen ausführbaren Dateien mit den Namen `S*` (S wie Start) aufruft.[5] Beim Verlassen des Runlevel 5 werden später alle dort vorhandenen ausführbaren Dateien namens `K*` (K wie Kill) aufgerufen. Der Vorteil liegt darin, dass nicht

[4] Bei manchen Systemen kann man auch `init` statt `telinit` eingeben, da das eine nur ein Link auf das andere ist.

[5] Bei manchen Distributionen ist es stattdessen das Verzeichnis `/etc/rc$1.d`. Das ist aber nur Formsache.

nur ein Start-, sondern auch ein Kill-Skript möglich insd. Der Nachteil ist, dass eine respawn-Überwachung so nicht möglich ist. Wenn ein Prozess dauerhaft laufen soll, muss der Wiederanlauf z.B. durch crontab sichergestellt werden.

Welche ausführbaren Dateien liegen nun in `/etc/init.d/rc${1}.d`? Bei der Installation eines Programms, das im Hintergrund laufen soll und mit Startskripten gestartet werden soll, wird in der Regel außer der Binärdatei noch ein Startskript mitgeliefert und im Verzeichnis `/etc/init.d` abgelegt. So gibt es zum Scanner-Server SANE die Binärdatei `/usr/sbin/saned` und das Startskript `/etc/init.d/saned`, das seinerseits die Umgebung setzt und die Binärdatei aufruft.

Terminal

```
root@debian964:~# /etc/init.d/saned start   # startet saned aktuell
root@debian964:~# /etc/init.d/saned stop    # stoppt saned aktuell
root@debian964:~# /etc/init.d/saned reload  # lädt neue Konfiguration
root@debian964:~# /etc/init.d/saned restart # Neustart von saned
```

Möchte man nun, dass der Scanner-Server im Runlevel 2 gestartet wird, legt man einen symbolischen Link vom Skript `/etc/init.d/saned` auf den neuen Namen `/etc/init.d/rc2.d/S47saned`.

Terminal

```
root@debian964:~# ln -s /etc/init.d/saned /etc/init.d/rc2.d/S47saned
```

Möchte man das nicht mehr, löscht man den Link wieder:

Terminal

```
root@debian964:~# rm /etc/init.d/rc2.d/S47saned
```

Der Buchstabe `S` steht für Start. Die Zahl `47` bewirkt, dass vor diesem Dienst alle Dienste mit niedrigeren Nummern und danach alle Dienste mit höheren Nummern liegen. Der Scanner-Server kann sich also auf darauf verlassen, dass alle Dienste mit niedrigerer Nummer schon fertig laufen, wenn sein Skript startet. Umgekehrt können sich alle folgenden Dienste darauf verlassen, dass der Scanner-Server schon fertig läuft. Der abschließende Namensbestandteil `saned` dient nur zur Orientierung für den Administrator.

3.4.2.3 Besonderheiten bei einzelnen Distributionen Bei einzelnen Distributionen mit Sys-V-Init gibt es ein paar Besonderheiten.

a) Bei Debian-Versionen mit Sys-V-Init sind die Runlevel 3 bis 5 nicht in Benutzung. Runlevel 2 ist dort bereits die Betriebsart mit Netzwerk und GUI.

b) Bei Debian-Versionen mit Sys-V-Init liegen die Verzeichnisse `rc0.d` bis `rc6.d` nicht in `/etc/init.d/`, sondern direkt in `/etc`.

c) Bei Distributionen, die der LSB (Linux Standard Base) folgen (das sind z. B. SuSE und Debian), enthält jedes Init-Skript in seinen ersten, auskommentierten Zeilen zwischen den Worten `BEGIN INIT` und `END INIT` Informationen über Abhängigkeiten, Standard-Einstellungen zu Start oder Nicht-Start in bestimmten Runleveln.

   Mit den Programmen `update-rc.d` (Debian) oder `insserv` (andere) kann man nun – in Abhängigkeit von den dortigen Informationen – den Dienst für bestimmte Runlevel freigeben oder sperren:

   Terminal

   ```
   root@debian964:~# update-rc.d dnsmasq defaults # Links anlegen
                                                  # nach Std.-Einstell.
   root@debian964:~# update-rc.d saned remove  # alle Links loeschen
   root@debian964:~# update-rc.d atd enable 2 3 # Links in rc2.d
                                                  # und rc3.d anlegen
   root@debian964:~# update-rc.d ssh enable 4 5 # Links in rc4.d
                                                  # und rc5.d entfernen
   ```

   Das Paket `sysv-rc-config` bietet eine ähnliche Funktionalität.

3.4.2.4 Besonderheit `rc.local` Es gibt bei den meisten Linux-Distributionen eine Stelle, an der man ohne großen Aufwand eigene kleinere Modifikationen anbringen kann. Dazu eignet sich die Datei `/etc/rc.local`, die bei Sys-V-Init gegen Ende des Startvorgangs von `/etc/init.d/rc.local` aufgerufen wird. Normalerweise sieht sie etwa so aus (hier ein gekürztes Beispiel aus Debian):

```
1 #!/bin/sh -e
2 # This script is executed at the end of each multiuser runlevel.
3 # Make sure that the script will "exit 0" on success or any other
4 # value on error.
5 # By default this script does nothing.
6 /usr/bin/rdate -n 192.53.103.103 &
7 exit 0
```

Auch hier ist zu beachten (wie bei Prozessen, die mit `/etc/inittab` gestartet werden), welche Shell verwendet wird (hier: `sh`!) und welche Umgebungsvariablen bekannt sind (am besten davon ausgehen, dass es keine sind).

3.4.3 Systemd

3.4.3.1 Allgemeines Beim Systemd-Init tritt an die Stelle der vielen Shell-Skripte ein einzelnes Programm, das in jedem Moment entscheidet, welche Prozesse gestartet und welche beendet werden müssen. In dieser Hinsicht ist es durchaus vergleichbar mit der Konfiguration durch `/etc/inittab`.

Ein großer Unterschied liegt darin, dass die Konfiguration verteilt ist auf viele kleine Textdateien, die klassischen ini-Dateien aus der Welt der Windows-PCs ähneln. Diese Konfigurationsdateien heißen hier *units*. Für jeden Prozess, der im Hintergrund laufen soll, wird eine Konfigurationsdatei geschrieben und im richtigen Verzeichnis abgelegt.

Ein anderer Unterschied zum inittab-System besteht darin, dass man in den units Abhängigkeiten zwischen verschiedenen Prozessen beschreiben kann. So soll etwa ein Webserver erst dann hochlaufen, wenn die Netzwerkschnittstellen in Betrieb sind.

Bei den rc.d-Startskripten berücksichtigt man solche Abhängigkeiten auch, aber nur dadurch, dass alle Skripte nacheinander durchlaufen müssen. Bei Systemd-Init dagegen dürfen Startvorgänge auch parallel ablaufen.

3.4.3.2 Orte für Units Wo findet nun die Konfiguration von Systemd statt? Die offizielle Dokumentation gibt folgende Reihenfolge an:

a) Optionen vom Bootloader

b) Dateien in `/etc/systemd/*.conf`, z. B. `system.conf`

c) Die oben genannten Units

Bei systemd werden also die (meisten) Informationen zur Konfiguration in kleinen Konfigurations-Textdateien gespeichert. Sie heißen *Units* und liegen z. B. im Verzeichnis `/lib/systemd/system/`. Die entsprechenden symbolischen Links findet man im Verzeichnis `/etc/systemd/system/`. Mit den Links kann man bestimmen, welche Unit nun wirksam sein soll und welche nicht. Tabelle 2 zeigt die verschiedenen Ebenen und Orte. Die Ebenen 3 und 4 beziehen sich auf Dienste, die nach Login eines Nutzers gestartet werden.

3.4.3.3 Format der Units Das Format der Unit-Dateien ist recht einfach gehalten:

```
1 [Kuchen]
2 Temperatur=180
```

Nr.	Ort	Reichweite	Art	änderbar
1	`/lib/systemd/system/*`	systemweit	installierbare	nein
2	`/etc/systemd/system/*`	systemweit	installierte	ja, durch Link auf 1
3	`/usr/lib/systemd/*/*`	jeder Nutzer	installierbare	nein
4	`ls /etc/systemd/user/*`	jeder Nutzer	installierte	ja, durch Link auf 3

Tabelle 2: Ebenen und Orte für Units

Eine Zeile mit einem Wort in eckigen Klammern leitet einen Abschnitt ein (genannt *Sektion*), wobei das Wort die Überschrift ist. Eine Zeile mit einem Gleichheitszeichen weist einem Schlüssel (einer Variablen) einen Inhalt (einen Wert) zu. Solch eine Zeile heißt *Direktive*.

Units haben in der Regel drei Sektionen, nämlich eine mit dem Namen *Unit*, eine, deren Name der Art der Unit (z. B. Service) ist und eine mit dem Namen *Install*:

```
1 [Unit]                        # Abschnitt: ueber diese Unit
2 Description=ErreichbarWeb     # Dir.: Beschreibung
3
4 [Service]                     # Es handelt sich um Service-Unit
5 Type=Simple                   # Dir.: vgl. respawn
6 ExecStart=/usr/local/bin/erreichbar.sh # Dir.: Start-Bin.
7
8 [Install]                     # Wann benutzen?
9 WantedBy=multi-user.target # Bei diesem Runlevel
```

3.4.3.4 Arten von Units Es gibt verschiedene Arten von Units. Zwei davon sind besonders wichtig, nämlich Targets und Services.

Eine *Target*-Unit dient zur Festlegung einer Betriebsart, ähnlich den Runlevel bei Sys-V-Init. Allgemein werden in einer Target-Unit die Beziehungen zu einer Gruppe anderer Units beschrieben. Meistens geht es darum, welche anderen Units zuerst beachtet werden müssen, bevor die eigene Unit fertig aufgebaut ist.

Die andere Art ist die *Service*-Unit. Sie dient meistens dazu, einen Systemdienst zu starten, zu stoppen und zu verwalten. In einer Service-Unit wird beschrieben, was gestartet werden soll und welche Voraussetzungen dafür erforderlich sind.

Außerdem gibt es noch weitere Unit-Arten, die in Tabelle 3 aufgeführt sind..

Name	Bedeutung
target	Runlevel
service	Systemdienst start,stop,Verwaltung
device	Anlegen einer Gerätedatei
mount	Partition ein-/aushängen
network	Netzwerk betreffend
path	Beobachtung Dateisystemobjekt
socket	Prozesse verbinden
timer	Zeitsteuerung (vgl. cron, anacron)

Tabelle 3: Arten von Units (Auswahl)

3.4.3.5 Wichtige Targets Es gibt spezielle Units, die eine bestimmte Betriebsart verkörpern und damit den Runleveln in Sys-V-Init entsprechen. Sie heißen *Targets*. Tabelle 4 zeigt, welches Target ungefähr welchem Runlevel entspricht. Das Default-Target wird durch einen symbolischen Link mit dem Namen `default.target` erzeugt, der auf eines der in der Tabelle genannten Targets zeigt.

Runlevel	Target
0	halt.target, poweroff.target
1	rescue.target, basic.target, ctrl-alt-del.target
2	multi-user.target
3	multi-user.target
5	graphical.target
6	reboot.target

Tabelle 4: Runlevel und Targets

3.4.3.6 Beispiel einer Service-Unit Eine Service-Unit entspricht in etwa einem Startskript und gibt an, wie und wann ein bestimmter Prozess gestartet werden soll. Hier ist ein Beispiel einer Unit (aus Debian) zu sehen:

```
1  [Unit]
2  Description=Scanner Service
3  Requires=saned.socket
4
5  [Service]
6  ExecStart=/usr/sbin/saned
7  User=saned
8  Group=saned
9  StandardInput=null
10 StandardOutput=syslog
11 StandardError=syslog
12 Environment=SANE_CONFIG_DIR=/etc/sane.d
13 Type=simple
14
15 [Install]
16 Also=saned.socket
```

Zeile 1 Die Zeilen mit den eckigen Klammern beginnen einen Abschnitt in der Konfigurationsdatei. Die gezeigten drei Abschnitte sind üblich.

Zeile 3 Hier geht es um Abhängigkeiten im Betrieb der Unit. `saned` braucht zum Betrieb `saned.socket`. Andere mögliche Variablen sind z. B. `After`, `Before` und `Wants`.

Zeile 6 Hier ist die zu startende Befehlszeile.

Zeile 7 Benutzerkennung, unter dem das Programm starten soll

Zeile 8 Gruppenkennung, unter der das Programm starten soll

Zeile 12 Liste von Umgebungsvariablen, die für das Programm gesetzt sein sollen

Zeile 13 Vergleichbar mit dem Feld Aktion in der `inittab`. Andere mögliche Werte sind `forking`, `notify` und `oneshot`.

Zeile 16 Hier geht es um Abhängigkeiten bei der Installation: Was ist noch zu installieren. Eine andere mögliche Variable wäre `WantedBy`.

Weitere Informationen zu Service-Units erhält man mit `man systemd.service`. Wenn man wissen will, welche Arten von Units es noch gibt, kann man `man systemd.unit` durchsehen.

Direktive	Bedeutung
Type	Art des Dienstes (meist: simple)
ExecStart	Befehlszeile zum Start
ExecStartPre	Optional: Befehl vor dem Start
ExecStartPost	Optional: Befehl nach dem Start
WorkingDirectory	Optional: Aktuelles Verzeichnis
User	Optional: Eigentümer (default: root)
Group	Optional: Gruppe

Tabelle 5: Service-Direktiven

3.4.3.7 Wichtige Service-Direktiven und Service-Types Tabelle 5 zeigt eine Übersicht der wichtigsten Service-Direktiven. Von diesen Direktiven ist die Type-Direktive hervorzuheben. Sie gibt den Typ des Services an; der sagt aus, wie oft der Service gestartet werden soll (Tabelle 6).

Type	Bedeutung
simple	nach Ende sofort wieder starten (=respawn)
oneshot	einmal starten (=once)

Tabelle 6: Service-Typ

3.4.3.8 Verwaltung eines Services mit `systemctl` Zur Steuerung von Systemd dient das Programm `systemctl`. Es wird aufgerufen in folgender Weise:

Terminal
```
systemctl Aktion Unit
```

Oft benutzte Aktionen sind:

a) start, stop, restart, reload, status und kill – sie beziehen sich auf den aktuellen Zustand einer Unit

b) enable, disable und is-enabled – sie beziehen sich darauf, ob die Unit automatisch geladen wird

c) mask und unmask – sie verhindern, dass die Unit per Abhängigkeit geladen wird

d) edit – damit kann die Unit (ohne Angabe eines Pfadnamens) editiert werden

Hier einige Beispiele:

Terminal
```
root@debian964:~# systemctl start sshd  # startet ssshd
root@debian964:~# systemctl stop sshd   # stoppt ssshd
root@debian964:~# systemctl reload sshd # laedt ssshd mit neuer Konf.
root@debian964:~# systemctl restart sshd# Neustart sshd
root@debian964:~# systemctl status sshd # Grobe Statusanzeige
root@debian964:~# systemctl show sshd   # Genaue Anzeige
root@debian964:~# systemctl enable sshd # dauerhaft freigeben
root@debian964:~# systemctl disable sshd # dauerhaft deaktivieren
root@debian964:~# systemctl isolate rescue.target # Runlevel wechseln
```

Eine Übersicht über gestartete Units gibt das Programm `systemd-clgs`.

3.4.3.9 Steuerung des Systems mit `systemctl`

Oft benutzte Aktionen sind:

a) halt, poweroff, rescue – zum Herunterfahren des Systems

b) isolate – zum Wechseln der Betriebsart (des Runlevels)

Mit der folgenden Befehlszeile wechselt man z. B. in das Target multi-user:

Terminal
```
root@debian964:~# systemctl isolate multi-user
```

3.4.4 Welche Startmethode ist bei meinem System aktiv?

Es ist nicht ganz einfach, herauszufinden, welche Startmethode gerade aktiv ist. Eine Möglichkeit ist, nachzusehen, welche Verzeichnisse man im Verzeichnisbau hat:

a) `/etc/init.d/` weist auf Sys-V-Init hin.

b) `/etc/systemd/` weist auf Systemd hin.

Bei mancher Distributionen kann es aber sein, dass sowohl Sys-V-Init als auch Systemd dateimäßig installiert sind; die Auswahl findet dann per Bootloader-Option statt[6].

Dann kommt die zweite Möglichkeit in Betracht: Man schaut nach, welche Prozesse laufen:

Terminal
```
schueler@debian964:~$ ps ax | grep "^ *1 "
  1 ?    Ss   0:03 /sbin/init
schueler@debian964:~$ pstree | head -1
systemd-+-ModemManager-+-gdbus
schueler@debian964:~$ top
...
  1 root   20  0  27372  6524  5104 S  0,0  0,3  0:03.59 systemd
...
```

Ein Blick ins Dateisystem zeigt, was `/sbin/init` ist:

Terminal
```
root@debian964:~# ls -l $(which init)
lrwxrwxrwx 1 root root 8 Jun 5 2024 /sbin/init -> /lib/systemd/systemd
```

`init` ist hier also nur ein Link auf `systemd`, und die aktive Startmethode ist Systemd.

[6] Wenn man dann z. B. mit Systemd starten will, bekommt die Kernelzeile im Bootloader GRUB die zusätzliche Otion `init=/bin/systemd`.