Aus dem Nähkästchen geplaudert: Der Sys-V-Init-Prozess

Startaufstellung

Beim Booten startet ein typisches Linux-System zahlreiche Hilfsprogramme. Sie konfigurieren das System und initialisieren Dienste, die fortan im Hintergrund laufen. Init fungiert als Mutter all dieser Prozesse und ist bei den meisten Linux-Distributionen und Unix-Systemen ähnlich aufgebaut. Marc André Selig

Wer unmittelbar nach dem Booten seines Unix-Computers an der Konsole das Kommando »ps ax« eingibt, wird mit einer Fülle laufender Programme konfrontiert. Von mehr oder weniger geheimnisvollen Kernelprozessen über die vielen Daemons mit ihren meist auf d endenden Namen bis zu den »getty«-Prozessen für die virtuellen Konsolen und »xdm«, »gdm« oder »kdm« für den grafischen Login – es ist alles vertreten.

Fragt sich, wer diese Prozessmeute auf das System hetzt, sie koordiniert und den Überblick behält. Der frisch gebootete Kernel ruft zunächst nur ein einziges Programm auf und dies initialisiert dann das System. Es heißt passenderweise »init« und übernimmt, von wenigen Ausnahmen abgesehen, den Start der übrigen Programme.

Welche Programme der Init-Prozess wann, wie und wie oft startet, hängt vom Administrator des Systems ab oder vom Hersteller der gerade benutzten Linux-Distribution. Niemand muss jedoch das »init«-Programm austauschen, es lässt sich sehr frei konfigurieren.

Einer für alles

Traditionell kommen unter Linux zwei verschiedene Init-Varianten zum Einsatz. Die erste heißt Simple-Init und besticht – wie der Name andeutet – durch ihre einfache Anwendung. Im Prinzip arbeitet Simple-Init nur ein langes Shellskript ab. Das startet nacheinander alle Programme, die für den Betrieb des Systems nötig sind. Das Skript liest bei Bedarf auch Konfigurationsdateien oder erstellt sie, anschließend bringt es die Daemons auf ihren Weg.

Obwohl Simple-Init durch seine einfache Eleganz besticht, wird es in typischen Linux-Distributionen schnell zum Klotz am Bein. Da es die gesamte Information über den Systemstatus in einer einzigen Datei zusammenfasst, haben die heutigen paketorientierten Distributionen damit ihre Schwierigkeiten. Jedes Mal, wenn der Benutzer ein Paket neu installiert oder vom System entfernt, muss der Paketmanager diese zentrale Skriptdatei bearbeiten. Dabei soll er auch noch zuverlässig die eigenen Änderungen des Admins berücksichtigen. Das ist gefährliches Terrain, da Shellskripte sehr komplex ausfallen können.

Fast jede Distribution setzt heute daher ein so genanntes Sys-V-Init ein. Dieses Verfahren ist nach dem Vorbild von Unix System V modelliert. Es setzt eine relativ komplizierte Hierarchie an Konfigurations- und Skriptdateien zusammen mit Symlinks ein. Damit modelliert es verschiedene Systemzustände – sowohl mit Rücksicht auf wahlfrei installierbare Software als auch auf unterschiedliche Betriebsmodi.

Viele kleine Helfer

Beispielsweise unterscheidet ein Sys-V-Init zwischen einem Einzelbenutzermodus, der für Wartungsarbeiten dient, einem Workstation-Modus ohne Netzwerkaktivität und einem Modus mit allen Diensten. Meist unterscheidet das System auch noch zwischen einer Login-Variante mit und einer ohne grafischer Oberfläche. Damit funktioniert der Login auch bei nur teilweise eingerichteten Neuinstallationen oder falsch konfigurierten Grafikkarten.

Diese unterschiedlichen Systemzustände heißen Runlevel, sie werden durch eine einstellige Nummer oder einen Buchstaben gekennzeichnet. Einer weit verbreiteten Konvention folgend steht Runlevel 0 für das Ausschalten des Computers, Runlevel 6 für einen Neustart. Runlevel

Manchmal ist alles anders

Dieser Artikel beschreibt die grundlegende Vorgehensweise der Runlevel-Konfiguration eines Unix-Systems. Egal ob Linux, eine BSD-Variante, Solaris, HP-UX oder andere Unix-Derivate – der Sys-V-Init-Prozess funktioniert immer gleich oder sehr ähnlich.
Einige Linux-Distributionen halten sich aber nicht ganz an diese Konvention. Die besonders in Deutschland beliebte Suse beispielsweise wird – so warnt uns ein Suse-Mitarbeiter – bei den Konfigurationsschritten dazwischenfunken und Teile der Einstellungen rückgängig machen oder verändern.
Wer auf einem Suse-System ein neues Init-Skript anlegen muss, sollte am besten die Hinweise beachten, die Suse in »/etc/init.d/skeleton« abgelegt hat. Ein weiterer Artikel zu diesem Thema wird wichtige Eigenheiten und Besonderheiten der verbreiteten Distributionen unter die Lupe nehmen.

1 (auch s oder S genannt) aktiviert den so genannten Single User Mode, bei dem Netzwerkaktivitäten und Benutzerlogins eingeschränkt oder ganz ausgeschaltet sind. Das System steht dann ausschließlich dem Nutzer Root für Wartungsarbeiten zur Verfügung.
Die Runlevel 2 bis 5 sind frei konfigurierbar und bei unterschiedlichen Distributionen auch unterschiedlich belegt. Als Faustregel gilt, dass ein höherer Runlevel meist mit einer größeren Featuritis einhergeht. Wegen seiner Verbreitung konzentriert sich dieser Artikel im Folgenden auf den Sys-V-Init-Prozess.

Zentrale Konfiguration

Die erste Konfigurationsdatei von »init« heißt »/etc/inittab«. Das Programm liest sie unmittelbar nach dem Systemstart, der Inhalt entscheidet über das weitere Vorgehen. Die wichtigsten Elemente der Inittab sind in Listing 1 abgedruckt. Jede Konfigurationszeile besteht aus vier Elementen, die durch Doppelpunkte getrennt sind. Dieses Prinzip ist bei älteren Konfigurationsdateien weit verbreitet. So findet es sich beispielsweise auch bei der Passwd-Datei oder der klassischen Printcap-Druckerdatenbank.

An erster Stelle einer Inittab-Zeile steht eine kurze Bezeichnung für das Programm. Sie darf maximal vier Zeichen lang sein, üblich sind ein bis zwei. Was dort genau steht, ist der Phantasie des Admin überlassen, seine Wahl sollte aber eindeutig sein. Das zweite Datenfeld gibt an, in welchen Runlevels dieser Dienst laufen soll. Das Beispiel startet in Zeile 12 bis 18 für jeden der Runlevel 0 bis 6 genau ein »rc«-Skript mit passenden Parametern. Der Shutdown-Befehl in Zeile 21 steht in jedem der Runlevel 1 bis 5 zur Verfügung.
Der dritte Eintrag nennt eine Aktion, die beschreibt, wann und wie der Init-Prozess das Kommando aus dieser Zeile ausführt. Meist ist als Aktion »wait« oder »respawn« angegeben.

Immer wieder starten

Bei »wait« startet Init das im vierten Feld angegebene Programm nur ein Mal, bei »respawn« hingegen immer wieder, sobald es sich beendet. Wait dient vor allem der Initialisierung des Systems, während Respawn-Einträge für fortlaufende Dienste stehen. Selbst wenn ein Daemon abstürzen sollten, startet ihn das System sofort neu. ►

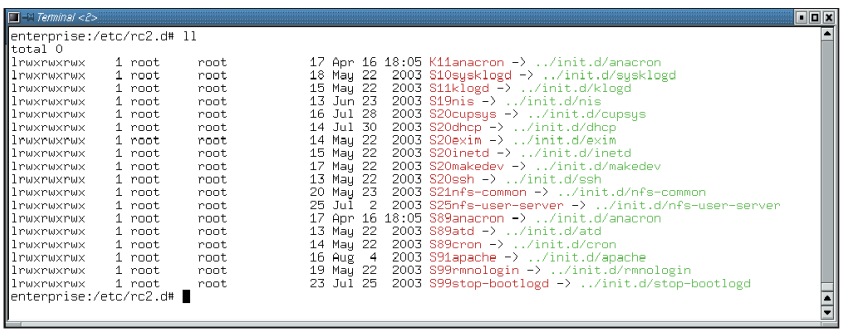

Abbildung 1: Das Verzeichnis »/etc/rc2.d« eines Debian-Systems enthält symbolische Links zu den Startskripten für Runlevel 2. Der Name des Links (rot) entscheidet, welches Skript »rc« wann und mit welcher Option aufruft. Die Startskripte (grün) liegen an zentraler Stelle in »/etc/init.d«.

Die Inittab in Listing 1 zeigt auch ein paar besondere Aktionen: In Zeile 2 gibt der »initdefault«-Eintrag den Runlevel an, den das System nach dem Booten als Erstes starten soll. Den »sysinit«-Eintrag (Zeile 6) führt Init immer zuerst aus, danach kämen die Einträge mit »boot« (alle parallel) und »bootwait« (einer nach dem anderen) an die Reihe. Für Sonderfälle gibt es noch weitere Aktionen, die beispielsweise auf die Meldungen einer unterbrechungsfreien Stromversorgung eingehen.

Listing 1: Init-Steuerungsdatei

```
01 # Default-Runlevel
02 id:2:initdefault:
03
04 # Skript, das beim Booten das System
05 # initialisiert und konfiguriert
06 si::sysinit:/etc/init.d/rcS
07
08 # Single-User-Modus
09 ~~:S:wait:/sbin/sulogin
10
11 # Die rc-Skripte der Runlevel
12 l0:0:wait:/etc/init.d/rc 0
13 l1:1:wait:/etc/init.d/rc 1
14 l2:2:wait:/etc/init.d/rc 2
15 l3:3:wait:/etc/init.d/rc 3
16 l4:4:wait:/etc/init.d/rc 4
17 l5:5:wait:/etc/init.d/rc 5
18 l6:6:wait:/etc/init.d/rc 6
19
20 # Auf [Ctrl]+[Alt]+[Delete] reagieren
21 ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -h now
22
23 # Getty-Prozesse
24 1:2345:respawn:/sbin/getty 38400 tty1
25 2:23:respawn:/sbin/getty 38400 tty2
26 3:23:respawn:/sbin/getty 38400 tty3
27 4:23:respawn:/sbin/getty 38400 tty4
28 5:23:respawn:/sbin/getty 38400 tty5
29 6:23:respawn:/sbin/getty 38400 tty6
```

Listing 2: Ein minimales Init-Skript

```
01 #! /bin/sh
02
03 test -x /usr/sbin/sshd || exit 0
04
05 set -e
06 case "$1" in
07   start)
08     echo -n "Starting sshd"
09     /usr/sbin/sshd
10     echo "."
11     ;;
12   stop)
13     echo -n "Stopping sshd"
14     kill `cat /var/run/sshd.pid`
15     echo "."
16     ;;
17 esac
18
19 exit 0
```

Gemäß Listing 1 hat der Init-Prozess nicht viel zu tun. Er liest »inittab« und stellt anhand der zweiten Zeile fest, dass er in Runlevel 2 wechseln soll. Bevor er das tut, startet er ein erstes Initialisierungsskript »rcS« und dann ein zweites namens »rc 2«, um in Runlevel 2 zu kommen. Außerdem ruft er ein paar »getty«-Prozesse ins Leben (Zeilen 24 bis 29). Diese zaubern die Login-Aufforderung auf die virtuellen Konsolen »tty1« bis »tty6«. Die Hilfsprogramme »rc« und »rcS« werden in HP-UX übrigens Sequencer-Skripte genannt, und zwar aus gutem Grund: Sie rufen die Startskripte in der gewünschten Reihenfolge auf.

Gelungene Arbeitsteilung

Die meiste Arbeit erledigen die »rc«-Skripte. Ihr Name ist frei wählbar und deshalb auch nicht in allen Distributionen gleich. Dabei ist ihr Verhalten im Großen und Ganzen identisch: Sie rufen die Startskripte in genau festgelegter Reihenfolge auf, die Vorgaben dazu befinden sich in bestimmten Verzeichnissen. Das Skript »rcS« sieht in »/etc/rc.S« nach, »rc 2« analog in »/etc/rc2.d«. Die Verzeichnisse verweisen auf Skripte (meist per Symlink oder Hardlink) mit bedeutungsschweren Namen. Abbildung 1 zeigt ein solches Verzeichnis auf einem Debian-System.

Der erste Buchstabe des Skript- oder Linknamens gibt an, ob der zugehörige Systemdienst gestartet (»S«) oder angehalten (gekillt, »K«) werden soll. Im Beispiel wird Init also »anacron« anhalten und die übrigen Dienste starten, darunter auch eine neue Anacron-Instanz. Auf den ersten Buchstaben folgen zwei Ziffern, die die Priorität bestimmen. Skripte mit niedriger Priorität laufen früher als solche mit höherer – die Zahlen geben die Sortierreihenfolge vor.

Der Rechner wird beim Wechsel in Runlevel 2 also zunächst einen eventuell laufenden Anacron-Prozess beenden. Danach ruft er »sysklogd« auf, anschließend »klogd« und »nis«. Der Syslog-Daemon soll möglichst von Anfang an laufen, um mögliche Fehlermeldungen aufzubewahren. Der Klog-Daemon benötigt einen laufenden Syslog und darf daher erst anschließend starten. Komplexe Dienste wie Apache kommen erst ganz zum Schluss an die Reihe, wenn das übrige System betriebsbereit ist.

Ein Init-Skript von innen

Init-Skripte folgen einer festen Konvention. Jedes Skript muss mindestens zwei Parameter verstehen: »start« und »stop«. Ist das Init-Skript in einem Runlevel-Verzeichnis mit einem Namen verlinkt, der mit »S« beginnt (im Beispiel etwa »S10sysklogd«), wird es vom »rc«-Skript für diesen Runlevel mit dem Parameter »start« aufgerufen. Beginnt sein Name dagegen mit »K« (wie bei »K11anacron«), lautet der Parameter »stop«.

Praktisch alle Init-Skripte werden also recycelt. Ein Skript sorgt ebenso für den Start eines Dienstes beim Booten wie für sein sauberes Ende während des Herunterfahrens. Das hilft den Überblick zu behalten: Ändert sich beispielsweise der Ort, an dem ein Daemon installiert ist, wird der Admin automatisch sowohl Start- als auch Stoppbefehle ändern. Aus diesem Grunde sind die Init-Skripte auch in einem eigenen Verzeichnis gesammelt, meist »/etc/init.d«. In den einzelnen Runlevel-Verzeichnissen liegen nur Links, die auf diese Originale verweisen. Es genügt also, das Skript in »/etc/init.d« anzupassen.

Listing 2 zeigt ein auf das Notwendigste reduziertes Startskript. Es prüft in Zeile 3, ob das Daemon-Programm verfügbar ist und nicht etwa gelöscht wurde. Anschließend ruft es – je nach Startparameter – den gewünschten Daemon auf oder beendet ihn. Zum Beenden dient ein »TERM«-Signal [1] an die Prozesskennung, die der Daemon selbst im Verzeichnis »/var/run« abgelegt hat.

Beim Installieren eines zusätzlichen Dienstes muss die Paketverwaltung dank Sys-V-Init keine vorhandenen Files ändern, sie kopiert nur das Startskript an den Zielort und legt die passenden Symlinks an. *(fjl)*

Infos

[1] Marc André Selig, „Handzeichen – Interprozesskommunikation mit Signalen“: Linux-Magazin 04/04, S. 76