Einführung Unix/Linux

Kapitel: Grundlegende Administration

Jens Roesen <jens@roesen.org>

Würzburg, März 2010
Version: 0.1.5 – **sehr beta** –

Dieses Dokument wurde in vim (`http://www.vim.org`) bzw. TEXnicCenter (`http://www.texniccenter.org/`) geschrieben und mit LaTeX (`http://www.latex-project.org/`) formatiert und gesetzt.
Die jeweils aktuelle Version ist unter `http://www.roesen.org` erhältlich.

Inhaltsverzeichnis

Vorwort

Motivation

Im Rahmen interner Schulungsmassnahmen kam die Frage nach geeigneten Schukungsmaterialien bzw. einem Skript für Unix-Neulinge auf. Schulungsunterlagen und Skripte für Einsteiger, aber letztendlich hat mir bei den meisten entweder etwas gefehlt, oder es war für unseren Zweck viel zu viel irrelevanter Stoff. Da wir in der Hauptsache mit Solaris und Linux-Systemen arbeiten und dabei Themen wie X-Windows oder Druckerverwaltung komplett ausklammern können, aber auf Themen wie Netzwerke, Troubleshooting, Mailserver oder DNS-Server Wert legen, habe ich mich irgendwann hingesetzt und angefangen dieses Skript zu schreiben.

Es ist der Versuch Systemadministratoren mit Grundkenntnissen in Unix den Arbeitsalltag zu erleichtern und dabei zwei verschiedene Unix-Geschmacksrichtungen, nämlich Solaris[1] und Red Hat Enterprise Linux, gleichermassen zu betrachten.

Mir ist durchaus klar, dass nicht alles im Folgenden beschriebene „state of the art“ ist bzw. sein kann und sicher auch noch etliche Fehler übersehen wurden. Wer einen solchen findet, weiss wie man einige Aufgaben besser lösen kann oder bessere Beispiele kennt ist herzlich eingeladen mir eine Mail an <jens@roesen.org> zu schicken.

Zielgruppe

Dieses Kurzscript ist als Crashkurs zur Einführung in die Administration von Unix und Linux Systemen gedacht. Es wird dabei ausschliesslich auf der Konsole und ohne grafische Oberfläche gearbeitet. Die gezeigten Beispiele beziehen sich auf Systeme unter Sun Solaris und RedHat Enterprise Linux.

Ohne Vorkenntnisse und Erfahrung mit Unix und/oder Linux Systemen wird der angesprochene Stoff teils nur schwer zu verstehen sein. Als alleiniges Lehrskript für blutige Anfänger ist es daher nicht geeignet obwohl in einigen Kapiteln vereinzelt kurz auf Grundlagen eingegangen wird (z.B. Kapitel 2).

Aufbau des Skripts

Wirr. Durch und durch. Aber zu mehr ist momentan keine Zeit. Ich habe versucht die Kapitel und Themen in eine halbwegs sinnvolle Reihenfolge zu bringen. Im Lauf der Zeit wird da sicherlich noch einiges umgestellt werden.

[1] In der vorliegenden Version des Skripts nur bis Version 9.

Typographisches

Da sich alle Beispiele, Kommandos und Ausgaben auf der Konsole abspielen, werden diese Bereiche entsprechend formatiert um sich vom regulären Text abzusetzen. Nach dem Login als User root, mit dem wir in diesem Skript hauptsächlich arbeiten werden, landet man in einem Command Prompt der so aussehen koennte:

```
[root@server1 root]#
```

Da dieser Prompt ja nach name des Systems oder aktuellem Verzeichnis mal kürzer aber auch sehr viel länger sein kann, wird der root Prompt auf

```
#
```

verkuerzt. Bitte den Hash (`#`) hier **nicht** als Kommentarcharakter verstehen, der unter Linux/Unix z.B. in Shellskripten die folgende Zeile vor der Ausführung durch die Shell schützt. Der normale User-Prompt, falls er uns wirklich einmal begegnen sollte, wird analog dazu auf

```
$
```

zusammengestrichen.

Für Konsolenausgaben, Konfigurationsdateien oder Teile von Skripten wird eine nichtproportionale Schrift verwendet:

```
if [ -n "$_INIT_NET_IF" -a "$_INIT_NET_STRATEGY" = "dhcp" ]; then
        /sbin/dhcpagent -a
fi
```

Werden in einem Beispiel Konsoleneingaben vom Benutzer erwartet, wird die in einer nichtproportionale Schrift dargestellt, wobei die Benutzereingaben fett gedruckt sind:

```
# uname -a
SunOS zoidberg 5.9 Generic_118558-21 sun4u sparc SUNW,Sun-Blade-100
```

Kommandos, Dateinamen oder Benutzerkennungen im laufenden Text werden ebenfalls in einer nichtproportionalen Schrift dargestellt: „Mit dem Befehl `pwd` kann überprüft werden, in welchem Verzeichnis man sich gerade befindet.“.

Müssen in einem Beispiel noch Teile der erwarteten Benutzereingaben durch die richtigen Werte ersetzt werden, so wird dieser Teil in kursiver Nichproportionalschrift dargestellt: „Für jedes Interface welches beim boot konfiguriert werden soll muß eine Datei **/etc/hostname.*interface*** existieren.“.

Eigennamen, Personen oder Organisationen erscheinen manchmal (ich bin gerade zu faul alle Vorkommen entsprechend zu formatieren) in Kapitälchen: „Eine sehr große Rolle hierbei hat die University of California in Berkley (UCB) gespielt, an der Thompson im Winter 76/77 eine Vorlesung zum Thema Unix abhielt.“.

1 Systemadministration

> LINUX: You will spend countless hours figuring out how to do the simplest things. What could be more fun?
>
> *(dumbentia.com)*

In diesem Kapitel will ich einige wenige wichtige Themen zur Systemadministration ansprechen. Das sind bei weitem nicht alle relevanten Gebiete der Systemadministration. Themen wie Kernel kompilieren oder Benutzerverwaltung fallen vorerst unter den Tisch.

1.1 Boot Prozess und init-System

Bootstrapping, von „To pull oneself up by one's bootstraps.“, bezeichnet den Vorgang den wir allgemein als Booten bezeichnen. Mittels eines einfachen Systems, wie dem BIOS oder später dem Boot-Loader, wird ein kompliziertes System gestartet. Der Ausdruck bootstrapping geht zurueck auf Baron Münchhausen, der sich selber samt Gaul an einen Schopf aus dem Morast zog, und die Science-Fiction Kurzgeschichte „By His Bootraps“ von Robert A. Heinlein.

Der Boot Prozess der meisten Unix Systeme läuft relativ ähnlich ab. Sie verwenden alle SysVinit, das init-System von Unix System V. Nur Ubuntu, Fedora seit Version 9 und der Experimental Branch von Debian nutzen statt SysVinit mittlerweile das für Ubuntu entwickelte upstart[1].

Nachdem bei Intel/x86 Systemen der POST[2] vom BIOS[3] abgeschlossen wurde, werden die angeschlossenen Devices nach Bootsektoren (z.B. dem MBR[4] durchsucht. Der im Bootsektor liegende Boot-Loader wird gestartet und dieser lädt das Kernel-Image. Der Kernel erkennt die Hardware, bindet die Laufwerke ein und startet schliesslich den init-Prozess.

Bei Sun Sparc Systemen kontrolliert der Boot-PROM[5] und der auf ihm gespeicherte Monitor namens OpenBoot das System bis der Solaris Kernel geladen werden kann. Der

[1] `http://upstart.ubuntu.com/`
[2] Power On Self-Test
[3] Basic Input Output System
[4] Master Boot Record, die ersten 512 Byte einer Festplatte mit Partitionstabelle und Boot-Loader.
[5] Programmable Read-Only Memory

Boot-PROM lädt den `bootblk`, dieser startet das Programm ufsboot welches schliesslich den Kernel einliest. Sobald der Kernel mit Hilfe von ufsboot genug Module geladen hat um das Root-FS zu mounten, startet den init-Prozess.Der init-Prozess, PID 1, sucht in der Datei /etc/inittab nach einem Eintrag für den default Runlevel.

```
id:3:initdefault:
```

Ein Runlevel in einem SysV-Unix-System beschreibt eine Art Systemzustand. Ein System kann so in einen definierten Zustand gebracht werden um z.B. spezielle Wartungsarbeiten im Single-User Mode durchzuführen. Es gibt normalerweise sieben Runlevel, 0 bis 6, von denen nur die Runlevel 0 (System-Halt), S (Single-User Mode, meist ein Synonym für Runlevel 1) und 6 (Reboot) auf allen SysV-Systemen identisch sind. Alle übrigen Runlevel werden je nach Distribution und Hersteller anders benutzt.

Einem Runlevel sind Dienste zugeordnet welche beim Booten nach und nach gestartet werden. Dabei werden die einzelnen Runlevel gegebenenfalls nacheinander komplett durchlaufen, bis das System seinen default Runlevel erreicht hat. Unter Solaris ist Runlevel 3 default[6]. Unter Linux ist es mit installierter graphischer Benutzeroberfläche Runlevel 5, ohne GUI ebenfalls Runlevel 3. Tabelle 1.1 stellt die Runlevel von Red Hat und Solaris gegenüber.

Red Hat und Solaris Runlevel		
Runlevel	Red Hat	Solaris
S	Single-User Mode, wie Runlevel 1	Single-User Mode, wie Runlevel 1
0	Halt/Shutdown	Shutdown, OpenBoot PROM Modus (`ok`)
1	Single-User Mode	Single-User Mode
2	Multiuser, einige Netzwerkservices werden gestartet	Multiuser ohne NFS Server
3	Multiuser mit Netzwerk	Multiuser mit NFS
4	nicht definiert, frei verfügbar	nicht definiert, frei verfügbar
5	Multiuser mit X11 und Netzwerk	Poweroff, ähnlich Runlevel 0 aber es wird noch sofern möglich das Netzteil abgeschaltet
6	Reboot	Reboot

Tabelle 1.1: Unix Runlevel

Die Funktion spezieller Runlevel wie Q, a, b, c oder U kann der Manpage zu `init` entnommen werden. In welchem Runlevel sich das System befinder kann man mit `runlevel` (Linux) oder `who -r` (Linux & Solaris) abfragen.

Wenn das System in den Runlevel 3 bootet, werden nacheinander die darunterliegenden Runlevel durchlaufen und die entsprechenden Startscripte/init-Scripte, all die mit

[6]Dies hat sich mit Solaris 10 geändert. Dort wurden „milestones“ eingeführt, über die Services gruppiert werden. „If services are files, then milestones are directories.“

einem S gefolgt von einer Zahl am Anfang des Dateinamens, der Reihe nach gestartet. Die init-scripte liegen unter Solaris alle in `/etc` bzw. `/etc/rc.*/` und `/etc/init.d/`[7]. Die Scripte aus den einzelnen Verzeichnissen (z.B. `/etc/rc3.d`) sind Hardlinks auf die Scripte in `/etc/init.d/` wie man am identischen Inode erkennt:

```
# ls -li /etc/rc3.d/S15nfs.server /etc/init.d/nfs.server
     788 -rwxr--r--   6 root     sys       2769 Apr  7  2002 /etc/init.d/nfs.server
     788 -rwxr--r--   6 root     sys       2769 Apr  7  2002 /etc/rc3.d/S15nfs.server
```

Unter Linux liegen die init-scripte unter `/etc/rc.d/rc.*/` bzw. `/etc/rc.d/init.d/`. Es gibt jedoch auch Hardlinks (RHEL 3) bzw. Symlinks (RHEL 5) von `/etc/rc.*` nach `/etc/rc.d/rc.*`, was die Umstellung von Solaris auf Linux erleichtert.

```
[RHEL3]# ls -ldi /etc/rc3.d/ /etc/rc.d/rc3.d/
5390340 drwxr-xr-x    2 root     root         4096 Jan 24  2007 /etc/rc3.d/
5390340 drwxr-xr-x    2 root     root         4096 Jan 24  2007 /etc/rc.d/rc3.d/

[RHEL5]# ls -ldi /etc/rc3.d/ /etc/rc.d/rc3.d/
655428 lrwxrwxrwx    1 root     root           10 Nov 3 11:38 /etc/rc3.d -> rc.d/rc3.d
655422 drwxr-xr-x    2 root     root         4096 Nov 3 12:03 /etc/rc.d/rc3.d
```

Alle diese init-Scripte können auch zum starten, stoppen oder reinitialisieren von Services genutzt werden. Dazu werden sie, je nach Bedarf mit einer der Optionen `start`, `stop`, `reload`, `status` oder `restart` aufgerufen. Welche Optionen unterstützt werden erfährt man in der Regel durch das Starten eines init-Scripts ohne Optionen.

```
# /etc/rc3.d/S80sendmail
Usage: /etc/rc3.d/S80sendmail start|stop|restart|condrestart|status
```

Nun bei Bedarf die passende Option anhängen und ausführen.

```
# /etc/rc3.d/S80sendmail restart
Shutting down sm-client:                                   [  OK  ]
Shutting down sendmail:                                    [  OK  ]
Starting sendmail:                                         [  OK  ]
Starting sm-client:                                        [  OK  ]
```

Über die `/etc/inittab` werden, besonders unter Linux, außer den Runleveln noch andere Einstellung vorgenommen. Die Virtuellen Consolen werden definiert, Tastenkombinationen wie CTRL-ALT-DEL werden abgefangen und Maßnahmen bei UPS powerfail bzw. powerrestore vorgegeben.

Zusätzlich zu den runlevelspezifischen Scripten in den `rc.*` Verzeichnissen gibt es unter Linux noch das Script `/etc/rc.local`. Dieses Script wird nach alles anderen init-Scripts aber noch vor dem Login-Prompt ausgeführt. Kommandos oder Dienste die in jedem Runlevel benötigt werden können auch in `rc.local` geschrieben werden.

[7] Seit Solaris 10 werden viele Services nicht mehr über die init-Scripte sondern über das Solaris Service Management Facility (`smf(5)`) gestartet.

1.2 Logging und Logfiles

Die verschiedenen Programme und Dienste eines Unix-Systems sind teilweise durchaus geschwätzig. Teilweise wollen sie einem nur mitteilen, dass sie genau das was wir von ihnen erwarten gemacht haben. Eine Mail zugestellt, eine Proxyanfrage beantwortet usw.. Viel wichtiger sind Meldungen und Warnungen über Probleme, fehlgeschlagene Aktionen oder gar kritische Fehler wie eine Kernel panic. All diese Meldungen werden in verschiedenen Logfiles protokolliert. Verantwortlich dafür ist der Syslog-Daemon `syslogd`.

Welche Meldungen `syslogd` protokolliert und wohin diese gespeichert werden, wird über die Datei `/etc/syslog.conf` konfiguriert. Die Meldungen werden dabei nach zwei Gesichtspunkten betrachtet: welcher Dienst hat die Meldung geschickt und wie wichtig ist sie. Man spricht von Facility und Priotity (auch level oder severity). Bei der Kombination aus `facility.priority` spricht man vom Selector. Das Ziel der Meldungen wird auch als action bezeichnet. Tabelle 1.2 listet alle vorhandenen Facilities auf und Tabelle 1.3 auf Seite 5 stellt die möglichen Priorities je nach Relevanz in absteigender Reihenfolge zusammen. Einträge in der `syslog.conf` haben folgendes Format:

```
facility.priority          destination
```

syslog Facilities	
facility	Bedeutung
`auth` bzw. `authpriv`	Security und Authorisationsmeldungen. Unter Linux wird `authpriv` statt `auth` verwendet
`cron`	Meldungen von cron und at (Siehe Kapitel 1.3
`daemon`	Meldungen von Daemons ohne eigene Kategorie
`ftp`	Meldungen des ftp-Daemons (nur Linux)
`kern`	Kernel-Meldungen
`lpr`	Medungen des Druckersystems
`mail`	Meldungen des Mailsystems
`news`	Meldungen Usenet
`syslog`	Interne Meldungen vom `syslogd`
`user`	Medlungen von User Prozessen, default facility
`uupc`	Unix to Unix Copy System
`local0-7`	frei verwendbar
*	alle facilities

Tabelle 1.2: syslog facilities

Pro Zeile kann man mehrere „Selectoren“ verwenden wenn diese durch ein „`;`“ getrennt werden[8]. Zusätzlich kann das Verhalten vom `syslogd` noch über das Asterisk (`*`), das Ausrufezeichen (`!`)und das Gleichheitszeichen (`=`) beeinflußt werden.

Ein einfaches Beispiel für einen `syslog.conf` Eintrag ist folgender:

[8]Hier das Semikolon nicht mit dem Komma verwechseln. Ein `kern.err,mail.warn` wird alle Kernel und Mail Meldungen ab dem Level `warn` loggen.

syslog Priorities	
priority	Bedeutung
`emerg`	Worst Case. System Panic. All hands abandon ship!
`alert`	Alarm der sofortiges Eingreifen erfordert
`crit`	kritischer Zustand eines Dienstes oder Systembestandteils
`err`	allgemeiner Fehler
`warning` bzw. `warn`	Warnungen, nicht gravierend aber auch nicht normal
`notice`	besondere Situation aber kein Fehler. Einträge mit priority `notice` müssen in einer extra Zeile erfolgen
`info`	normaler Betrieb
`debug`	Debug Meldungen
`none`	keinerlei Meldung von entsprechender facility protokollieren

Tabelle 1.3: syslog priorities

```
mail.info          /var/log/mail
```

Damit werden alle Meldung des Mailsystems mit der Priorität info und höher in die Datei `/var/log/mail` geschrieben. Sollen nur die Meldung von genau einer Priorität erfasst werden, wird die mit dem = angezeigt: `mail.=info`. Folgender Eintrag würde jede `crit` Meldungen aller Facilities nach `/var/log/critical` schreiben, ausgenommen alle Kernel-Meldungen:

```
*.=crit;kern.none          /var/log/critical
```

Um Meldung nur bis zu einer bestimmten Priorität zu erfassen, gibt nach die ersten nicht mehr zu loggende Priorität mit vorangestelltem `!` an. Im folgenden Beispiel werden alle Kernel-Meldungen bis zur Priorität `warning` nach `/var/adm/kernel-info` geschrieben, alle Meldungen ab `err` werden nicht mitgeschrieben.

```
kern.!err          /var/adm/kernel-info
```

Dabei darf man „`!`“ und „`=`“ auch zu einem „alles, ausser genau dieser Priorität“ kombinieren: `mail.!=info`.

Als Ziel der `syslog`-Meldungen können neben normalen Dateien auch named Pipes/FIFOs (gekennzeichnet durch ein vorangestelltes „`|`“), andere Hosts (gekennzeichnet durch ein „`@`“ vor dem Hostnamen), einzelne bzw. mehrere durch „,“ getrennte User oder auch alle eingeloggten User („`*`“) angegeben werden.

Die meisten Logfiles liegen unter `/var/log` (Linux, Solaris) bzw. `/var/adm` (Solaris).

1.3 Prozessmanagement

Die Überwachung von Prozessen und, zu einem gewissen Teil, auch die Beeinflussung der Prozessprioritäten gehören zum täglichen Handwerk eines Admins. Als Prozess verstehen wir im wesentlichen ein laufendes Programm ohne Betrachtung um was es sich dabei genau handelt.

1.3.1 Prozessattribute und Eigenschaften

Jeder Prozess verfügt über diverse Attribute, über die das Betriebssystem die diversen Prozesse verwaltet. Diese Attribute sind auch für uns als Admins sehr wichtig, um das Zusammenspiel der Prozesse untereinander zu verstehen.

Die Prozess ID (PID) eines jeden Prozesses ist einzigartig auf dem jeweiligen System. Es sollten auf einem System nie zwei unterschiedliche Prozesse mit derselben PID existieren. Da jeder Prozess durch einen anderen Prozess gestartet wurde, ist für jeden Child-Prozess auch die PID seines Parent-Process bekannt, die PPID. Wie Dateien gehoeren Prozesse auch immer einem User und einer Gruppe. Nur der Eigentümer eines Prozesses und der Superuser root darf die Prozesseigenschaften ändern oder ihn beenden. Die „Besitzverhältnisse“ werden in der Regel vom Parent an seine Child-Prozesse vererbt. All diese und noch weitere Attribute wie die Priotität mit der der Prozess Rechnzeit von der Prozessverwaltung, dem Scheduler, zugewiesen bekommt, in welcher Umgebung er läuft oder welchen Status er momentan innehat ist im Pseudifilesystem unter `/proc`[9] abgelegt. Für jeden laufenden Prozess gibt es unter `/proc` ein Verzeichnis `PID`.

```
# ls /proc/2194/
cmdline  cpu  cwd  environ  exe  fd  maps  mem  mounts  root  stat  statm  status
```

Normalerweise muß man aber selber in den `/proc/$PID` Verzeichnissen nichts ändern.

1.3.2 Prozesslisten abfragen

Alle Prozessinformationen können durch diverse Tools abgefragt werden. Das bekannteste davon ist `ps`, process status. Wird ps ohne Optionen aufgerufen zeigt es lediglich die Prozesse an, die demselben User gehören und auf demselben Terminal (`TTY`) wie `ps` gestartet wurden.

```
# ps
   PID TTY      TIME CMD
 27986 pts/2    0:00 bash
 28126 pts/2    0:00 ps
```

Bisschen mager. Die gebräuchlichsten Aufrufe sind unter Solaris `ps -afe` (full listing aller laufenden Prozesse, siehe Abbbildung 1.1 auf Seite 7) und unter Linux `ps aux` (alle laufenden Prozesse im user-oriented format, siehe Abbbildung 1.2 auf Seite 8). Es gibt etliche Möglichkeiten `ps` zu beeinflussen. Am besten man sieht sich die Manpage an und experimentiert etwas herum.

```
# ps -afe
     UID   PID  PPID  C    STIME TTY    TIME CMD
    root     0     0  0   Mar 04 ?      0:03 sched
    root     1     0  0   Mar 04 ?      0:21 /etc/init -
    root     2     0  0   Mar 04 ?      0:03 pageout
    root     3     0  0   Mar 04 ?     1532:16 fsflush
    root   338     1  0   Mar 04 ?      0:00 /usr/lib/saf/sac -t 300
    root   221     1  0   Mar 04 ?     14:43 /usr/sbin/nscd
 postfix  3964   454  0   Sep 30 ?      1:20 qmgr -l -t fifo -u
  nobody 14815   164  0   Mar 10 ?      0:01 fs
    root 23178   164  0   Feb 24 ?      0:00 in.tnamed
    root 29010   292  0 09:42:49 ?      0:01 /usr/sbin/sshd2
    root 14812   164  0   Mar 10 ?      0:01 rpc.metamedd
  drizzt 24714 24705  0   Mar 26 pts/6 63:43 /opt/Irssi/bin/irssi
    root 14817   164  0   Mar 10 ?      0:02 rpc.ttdbserverd
  [...]
```

Abbildung 1.1: Gekürzte Ausgabe von `ps -afe` unter Solaris

`ps` wird häufig in Kombination mit `grep` verwendet um nach Prozessen zu suchen. Wenn man eine Übersicht über die laufenden Prozesse haben will bietet sich das Tool `top` an. `top` ist nicht automatisch auf jeder Solaris Installation verfügbar. Seit Solaris 8 gibt es jedoch `prstat`[10] welches ebenfalls seinen Zweck erfüllt.

`top` zeigt nicht nur eine automatisch aktualisierte Prozessliste an, sondern auch Informationen über die CPU Auslastung, Prozessprioritäten, Speicherverbrauch, IOwait, die Uptime und Load des Systems usw.. Schickt man `top` ein `?` gelangt man zu einer kleinen Hilfeseite. Man kann direkt aus `top` heraus viele der alltäglichen Administrationsaufgaben laufende Prozesse betreffend erledigen. Es lassen ich Prozesse killen (`k`) oder neu priorisieren (`r`). Ebenso läßt sich die Darstellung anpassen. Sortierung nach Speicherverbrauch (`M`) oder Ausblenden bestimmer Informationen sind möglich.

1.3.3 Vorder- und Hintergrundprozesse

Laufende (interaktive) Prozesse können vom Anwender beliebig zwischen Vorder- und Hintergrund verschoben werden. Lang laufende `grep`-Suchen können so im Hintergrund weiterlaufen während man im Vordergrund weiter arbeitet. Man kann einen Prozess entweder direkt so starten, dass er direkt im Hintergrund läuft oder den laufenden Prozess in den Hintergrund verschieben.

Um einen Prozess direkt im Hintergrund zu starten hängt man einfach ein „Ampersand“ `&` an dem Befehl an.

```
# grep -i kernel /var/adm/messages &
```

[9] Manpage zu `proc` in der Sektion 5 bei Linux und Sektion 4 bei Solaris.

[10] Topping top in Solaris 8 with prstat, `http://developers.sun.com/solaris/articles/prstat.html`

```
# ps aux
USER     PID %CPU %MEM    VSZ   RSS TTY   STAT START   TIME COMMAND
root       1  0.0  0.1   2064   404 ?     Ss   08:12   0:00 init [5]
root       2  0.0  0.0      0     0 ?     S<   08:12   0:00 [migration/0]
root       4  0.0  0.0      0     0 ?     S<   08:12   0:00 [watchdog/0]
root       5  0.0  0.0      0     0 ?     S<   08:12   0:00 [events/0]
root      82  0.0  0.0      0     0 ?     S<   08:12   0:00 [cqueue/0]
root      85  0.0  0.0      0     0 ?     S<   08:12   0:00 [khubd]
root     379  0.0  0.0      0     0 ?     S<   08:13   0:00 [kauditd]
root     413  0.0  0.1   2992   284 ?     S<s  08:13   0:00 /sbin/udevd -d
root     690  0.0  0.0      0     0 ?     S<   08:13   0:00 [kgameportd]
root    4185  0.0  0.2  12048   484 ?     S<sl 08:13   0:00 /sbin/audispd
root    4201  0.0  0.1  10236   276 ?     Ss   08:13   0:00 /usr/sbin/restorecond
root    4212  0.0  0.2   1720   492 ?     Ss   08:13   0:00 syslogd -m 0
root    4215  0.0  0.1   1672   292 ?     Ss   08:13   0:00 klogd -x
root    4241  0.0  0.1   2156   380 ?     Ss   08:13   0:00 mcstransd
rpc     4254  0.0  0.1   1804   420 ?     Ss   08:13   0:00 portmap
root    4289  0.0  0.2   1920   620 ?     Ss   08:13   0:00 rpc.statd
root    4325  0.0  0.1   5432   316 ?     Ss   08:13   0:00 rpc.idmapd
dbus    4344  0.0  0.4  13120   916 ?     Ssl  08:13   0:00 dbus-daemon --system
root    4456  0.0  0.4  10832  1052 ?     Ssl  08:13   0:00 automount
root    4475  0.0  0.1   1672   368 ?     Ss   08:13   0:00 /usr/sbin/acpid
root    4506  0.0  0.3   6988   804 ?     Ss   08:13   0:00 /usr/sbin/sshd
[...]
```

Abbildung 1.2: Gekürzte Ausgabe von `ps aux` unter Linux

```
[1] 23016
```

Laufende Prozesse verschiebt man mit `CRTL-Z (^Z)` in den Hintergrund. Diese werden damit allerdings auch gestoppt und können mit `bg` wieder getsartet werden.

```
# some-process
^Z
[1]+  Stopped                 some-process
# bg
[1]+ some-process &
```

Welche Prozesse derzeit im Hintergrund laufen und welche Job-Nummer ihnen zugewiesen wurde kann man sich ueber das Kommando `jobs` anzeigen lassen.

```
# jobs
[1]-  Exit 1                  grep -i kernel /var/adm/messages
[2]+  Stopped                 vi hostname.qfe5
```

Die Ausgaben von `jobs` unterscheiden sich dabei je nach Shell etwas. Mit `fg %<`*job nummer*`>` kann ich mir einen Prozess wieder in den Vordergrund holen.

```
top - 11:23:27 up 22 days,  4:02,  2 users,  load average: 0.00, 0.00, 0.00
Tasks: 100 total,   1 running,  99 sleeping,   0 stopped,   0 zombie
Cpu0  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  0.3%us,  0.0%sy,  0.0%ni, 99.3%id,  0.3%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4025648k total,   657796k used,  3367852k free,   219540k buffers
Swap:  4883752k total,        0k used,  4883752k free,   248144k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
16724 drizzt    20   0  2308 1116  860 R    0  0.0   0:00.10 top ´
    1 root      20   0  2712 1736  612 S    0  0.0   0:01.98 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.01 migration/0
    4 root      15  -5     0    0    0 S    0  0.0   0:00.09 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/0
    6 root      RT  -5     0    0    0 S    0  0.0   0:00.01 migration/1
    7 root      15  -5     0    0    0 S    0  0.0   0:00.04 ksoftirqd/1
    8 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/1
    9 root      RT  -5     0    0    0 S    0  0.0   0:00.01 migration/2
   10 root      15  -5     0    0    0 S    0  0.0   0:00.10 ksoftirqd/2
   11 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/2
   12 root      RT  -5     0    0    0 S    0  0.0   0:00.01 migration/3
   13 root      15  -5     0    0    0 S    0  0.0   0:00.14 ksoftirqd/3
```

Abbildung 1.3: Gekürzte Ausgabe von `top` auf einer Mehrprozessormaschine

1.3.4 Prozesse beenden

Sollte einmal nötig einen einen Prozess außerplanmäßig beenden zu müssen kommen die Befehle `kill`, `killall`[11] und `pkill` ins Spiel. Diese drei Tools erlauben es dem Admin Signale[12] an Prozesse zu schicken und sie so zu beeinflussen. Dabei muss `kill` mit einer PID aufgerufen werden während sich `killall` und `pkill` auch mit Namen von Prozessen zufriedengeben. Ein einfaches `kill <PID>` sendet ein SIGTERM (termination signal) an den Prozess. Sollte das nicht ausreichen um den Prozess zu beenden, kann man mit `kill -9 <PID>` ein SIGKILL (kill signal) senden nachdem der Prozess beendet sein sollte. Mit einem SIGHUP Signal (hangup signal, `kill -1 <PID>`) kann man Daemons (z.B. `inetd` dazu veranlassen ihre Konfiguration neu einzulesen. `killall` und `pkill` arbeiten ähnlich, werden aber statt mit der PID mit dem Namen des zu killenden Prozesses aufgerufen. Wer gerne Fussnoten überliest weil da eh nur Mist drinsteht sei nochmal drauf hingewiesen, dass ein `killall` unter Solaris alle laufenden Prozelle killt. Ohne wenn und aber.

[11] **ACHTUNG!** `killall` unter Solaris stoppt alle aktiven Prozesse.

[12] Eine Liste der möglichen Signale findet man unter Solaris mit `man -s 3head signal` und unter Linux mit `man 7 signal`.

1.3.5 Prozesse priorisieren

Eher selten muss man die Priorität mit der ein Prozess Rechenzeit zugewiesen bekommt per Hand anpassen. Normalerweise weist der Scheduler den Prozessen abhängig von ihrer Priorität im round robin Verfahren Rechenzeit zu. Die Priorität wird durch eine ganze Zahl symbolisiert und ist umso höher je kleiner die Zahl ist. Pro Prozess gibt es zwei Prioritätswerte: die nice number stellt die erwünschte Priorität dar und die wirkliche aktuelle Priorität mit der der Prozess behandelt wird. Die nice number, `NI` in der Ausgabe von `ps` oder `top`, kann vom Prozesseigentümer und vom `root` angepasst werden. Die aktuelle Priorität wird im `ps` oder `top` unter `PRI` aufgeführt.

Die angestrebte nice number kann entweder beim Starten des Prozesses mit vorangestelltem `nice` direkt angeben oder nachträglich mit `renice` erbitten.

1.3.6 Zeitgesteuerte Prozesse mit `cron` und `at`

Unter Unix gibt es zwei Daemons, die einem Sysadmin erlauben den Start eines Prozesses an bestimmte Zeitvorganen zu knüpfen. Um bestimmte Prozesse in einer festen Zeitreihe ablaufen zu lassen vertraut man diese dem `cron`-System an. Soll ein Prozess nur einmalig zu einer vorgegebenen Zeit laufen bedient man sich des `at-Daemons.`

Das `cron` System

Viele Vorgänge und Aufgaben, wie wöchentliche Backups oder Statistiken über den Mailverkehr, müssen regelmäßig wiederholt werden. Alle diese Aufgaben kann man mit sogenannten Cronjobs zu vorgegebenen Zeiten automatsich starten. Dazu startet das System beim booten den Dienst `cron`. Wer letztendlich `cron` benutzen darf wird über die Dateien `/etc/cron.allow` und `/etc/cron.deny` geregelt. Unter Solaris liegen diese beiden Dateien in `/etc/cron.d/`. Existiert die `cron.allow` muss ein normaler User in ihr gelistet sein, um `cron` nutzen zu dürfen. Existiert sie nicht, darf er nicht in der `cron.deny` gelistet weis wenn er `cron` benutzen will. Existieren weder `cron.allow` noch `cron.deny` darf nur der Superuser `root` Cronjobs erstellen.

Cronjobs werden in `crontab`-Dateien gespeichert. Diese liegen unter `/var/spool/cron` (Linux) bzw. `/var/spool/cron/crontabs/` (Solaris). Man kann sich seine aktuelle `crontab`-Datei mit `crontab -l` anzeigen lassen oder sie mit `crontab -e` im $EDITOR bearbeiten. `crontab`-Dateien werden von `cron` zeilenweise gelesen. Jede Zeile stellt eine von `cron` durchzuführende Aufgabe dar und ist folgendermaßen aufgebaut:

```
minute      stunde      tag-im-monat      monat      wochentag      kommando
```

Die ersten fünf Felder definieren den Zeitpunkt an dem `kommando` ausgeführt werden soll. In Tabelle 1.4 ist aufgeführt welche Werte in die einzelnen Felder eingesetzt werden können.

Statt absoluter Werte können auch durch Komma getrennte Listen (`0,10,20,30`), Bereiche (`8-18`) und Schrittweiten mittels `/n` angegeben werden. Sehen wir uns ein paar Beispiele an.

crontab Zeitsteuerung		
Feld	Bedeutung	mögliche Werte
minuten	Minuten der Stunde	`0-59` und *
stunde	Stunde vom Tag	`0-23` (`0`=Mitternacht) und *
tag-im-monat	Monatstag numerisch	`1-31` und *
monat	Monat numerisch	`1-12` und *
wochentag	Wochentag numerisch	`0-6` (`0`=Sonntag) und *

Tabelle 1.4: crontab Zeitsteuerung

```
15     20     *     *     6    echo "‘Samstag, 20:15 Uhr."’
```

Dieser Job würde jeden Samstag (Eintrag `6` als Wochentag) umd 20:15 Uhr gestartet.

```
0,10,20,30,40,50  8-18  *  *  1-5  echo "‘Alle 10 Minuten, Wochentags, 8 bis 18 Uhr."’
```

Die Liste `0,10,20,30,40,50` für „alle vollen 10 Minuten“ könnte man auch als `0-59/10` bzw. `*/10` schreiben.

```
0  0  1,15  *  *  echo "‘Jeden 1. und 15. eines jeden Monats genau um Mitternacht."’
```

Bereiche und Listen lassen sich durch Komma getrennt ebenfalls kombinieren: `0,4,8-16,20` oder `8-12,14-18`.

Bei den auzuführenden Kommandos müssen Prozentzeichen „`%`“ falls vorhanden mit einem Backslash escaped werden, da sie sonst als Zeilentrenner interpretiert werden. Leerzeilen oder Zeilen beginnend mit „#“ (Kommentare) werden nicht ausgewertet.

Für Cronjobs werden automatisch einige Umgebungsvariablen gesetzt. Benutzername (`LOGNAME`) und Homedir (`HOME`) werden aus der `/etc/passwd` übernommen. Als default Shell wird unter Solaris die Bourne Shell verwendet (`SHELL=/bin/sh`), Linux übernimmt (je nach Distribution) auch diesen Wert aus der `passwd`. Der Pfad (`PATH`) ist fuer normaler User `/usr/bin`, für `root` hingegen `/usr/sbin:/usr/bin`. Cronjob `PATH` oder der der Shell unter scheiden sich also. Sollen für die durch `cron` ausgeführten Kommandos besondere Umgebungsvariablen gelten, so müssen diesen am Anfang der `crontab`-Datei bzw. im Kommandofeld durch `VARIABLE=WERT` gesetzt werden. Systemweit können diese Einstellungen unter Solaris über die Datei `/etc/default/cron` erfolgen.

Normalerweise werden alls durch `cron` gestarteten Job über `syslog` Protokolliert und die Ausgaben oder Fehlermeldungen an den Eigentümer des Jobs (`LOGNAME`) gemailt. Soll die Protokollierung über `syslog` ausbleiben, seltzt man ein „`-`“ vor das Kommando. Will man auf die gemailte Ausgabe verzichten, kann man `stdout` und `stderr` wie in Kapitel ?? angesprochen umleiten. Hier nocheinmal das Beispiel aus Kapitel ??.

```
1 2 * * * [ -x /usr/sbin/rtc ] && /usr/sbin/rtc -c > /dev/null 2>&1
```

Mittlerweile sollte jeder diesen Eintrag, abgesehen vielleicht vom `test`-Ausdruck `[ -x /usr/sbin/rtc ]`[13], komplett verstehen und beschreiben können.

[13]Mit `test` können Konditionen überprüft und zur Bedingung gemacht werden. Die `[...]` umschließen die zu testende Kondition. In diesem Fall mird mit `-x` geprüft, ob die Datei `/usr/bin/rtc` existiert und ausführbar ist.

Das at-System

Soll ein Job nur einmalig zu einer bestimmten Zeit laufen bietet sich als Scheduler der Dienst `at` an. Er wird ebenso wie `cron` beim booten gestartet. Analog zu `cron` wird über die Dateien `/etc/at.allow` und `/etc/at.deny` (Linux) bzw. `/etc/cron.d/at.allow` und `/etc/cron.d/at.deny`[14] festgelegt wer mit `at` arbeiten darf.

Als Argument verarbeitet `at` Zeitangaben, die durchaus komplex sein können. Tabelle 1.5 führt einige Beispiele auf.

Beispiele für `at` Zeitangaben	
Beispiel	Bedeutung
at now + 20 minutes	Job startet in 20 Minuten ab Eingabe.
at teatime Dec 24	Job startet am 24. Dezember um 16 Uhr
at 15:45 20.03.09	Job startet am 20. März 2009 um 15:45 Uhr
at 2 am ZULU + 3 days	Job startet um 2 AM nach ZULU (GMT) Zeit plus drei Tage.

Tabelle 1.5: crontab Zeitsteuerung

Wenn die Zeit in der Form „`HH:MM`“ angegeben wird, startet `at` den Job zum nächstmoglichen Zeitpunkt der auf `HH:MM` zutrifft. Wenn man um 16:10 Uhr einen Job für 16:00 Uhr einstellt wird dieser am nächsten Tag gestartet.

```
# date
Sun Nov 12 16:10:30 CET 2008
# at 16:00
at> echo foobar
at> <EOT>
job 4 at 2008-11-12 16:00
```

Ausser Zeitangaben in der Form `HH:MM` sind `now`, `teatime`, `noon` und `midnight` zulässig. Zusätzlich kann der Starttermin durch „`+ anzahl zeiteinheit`“ verschoben werden, wobei als Zeiteinheiten `minute(s)`, `hour(s)`, `day(s)` oder `week(s)` akzeptiert werden.

Nachdem man `at <zeit>` gestaretet hat, gelang man zu einem extra Prompt(„`at>` “) in dem na die auszuführenden Kommandos eingibt. Um diesen Eingabemodus zu verlassen und den `at`-Job zu speichern drückt man `CTRL-D` (`^D`). Alternativ können durch den Schalter `-f` auch Textdateien mit den auszuführenden Kommandos eingelesen werden. Danach gibt `at` die ID des Jobs sowie den Ausführungszeitpunkt aus. Den Stand bzw. die Anzahl der Jobs in der `at`-Queue zeigt der Befehl `atq` an. Mittels `atrm <job-id>` kann man einen Job löschen.

`at` speichert auch die wichtigsten Umgebungsvariablen, die zum Zeitpunkt der Joberstellung gültig waren[15].

Wie auch bei `cron` werden `stdout` und `stderr` der Kommandos aus dem `at`-Job per Mail dem Job-Eigentümer zugeschickt.

[14] Laut Manpage sollen die Dateien in `/usr/lib/cron/` liegen, allerdings ist `/usr/lib/cron/` ein Symlink nach `/etc/cron.d/`.

[15] Ausgenommen `TERM`, `DISPLAY`, _, `EUID` und `UID`