Bashscripting 104

„Forken ist gut, Kontrolle ist besser."

VON

MARIUS SCHWARZ

Bashscripting 104

ADVANCED BASHFUNCTIONS

Bashscripting 104

Die Aufgabe

Nehmen wir an, daß es einen Cron-Job gibt, der nachts einen Server aufräumen soll. Folgende Jobs wären zu erledigen:

1. /tmp/dynacache Verzeichnis aufräumen
2. Datenbank entmisten
3. Logfiles rotieren

Was wäre wohl der beste Weg, daß zu erledigen?

Bashscripting 104

So könnte das nötige Bashscript aussehen:

```
 !/bin/bash

rm -rf /tmp/dynacache/*
mysql -p09dJs22ss typo3datenbank -e „truncate t3_cache"
logrotate --log  /var/log/messages /etc/logrotate.syslogs
```

Bashscripting 104

DIE FRAGE LAUTET:

GEHT DAS VIELLEICHT EFFEKTIVER?

Bashscripting 104

DEFINIEREN WIR ERSTMAL, WAS „EFFEKTIVER" MEINEN KÖNNTE.

Bashscripting 104

1. Prozesse brauchen IO-Leistung der Festplatte

Bashscripting 104

1. Prozesse brauchen IO-Leistung der Festplatte
2. Prozesse brauchen CPU-Leistung (und RAM)

Bashscripting 104

1. Prozesse brauchen IO-Leistung der Festplatte

Alle Dateien in /tmp/dynacache zu löschen braucht

weder die CPU noch nennenswert RAM.

Bashscripting 104

2. Prozesse brauchen CPU-Leistung (und RAM)

Alle Einträge in einer Datenbanktabelle löschen, braucht anfangs kaum IO,

da der Datenbankserver die Tabellen im RAM vorhält,

wenn er kann, und (normalerweise) CPU Leistung.

Bashscripting 104

ERGO

KÖNNEN IO LASTIGE JOBS UND CPU INTENSIVE JOBS PARALLEL LAUFEN.

Bashscripting 104

Forken von Befehlen

Die Bash kann Programme starten, die „neben" dem Hauptprozess laufen und etwas tun.

Auf Deutsch nennt man das auch „absplitten" eines Prozesses, wobei das auch denglisch ist, genau wie „wegforken". Dafür gibt es mehrere Wege, der einfachste sieht so aus:

```
[marius eve   ]  find /home -name xorg.conf.backup  &
[1] 12719
[marius eve   ]
```

Der Anweisung, die man absplitten möchte, stellt man ein & ans Ende. Als Folge kann man direkt in der Shell weitermachen, da die Anweisung in einem eigenen Prozess läuft. Das lohnt sich für Aufgaben, die **sehr lange dauern** *und nicht beaufsichtigt werden müssen*.

Bashscripting 104

Angewandt auf unser Script, kann das so aussehen:

```
 !/bin/bash

rm -rf /tmp/dynacache/*
mysql -p09dJs22ss typo3datenbank -e „truncate t3_cache"
logrotate --log  /var/log/messages /etc/logrotate.syslogs
```

Nun werden alle Befehle „gleichzeitig" ablaufen. Da der crond das Script aufgerufen hat und alle abgesplitteten Prozesse die Ablaufkontrolle immer gleich wieder an die bash, die das Script ausführt, zurückgeben, ist das Ende des Script erreicht, bevor die abgesplitteten Prozesse fertig sind.

Bashscripting 104

DAS KÖNNTE UNTER UMSTÄNDEN NICHT DIE BESTE ENTSCHEIDUNG SEIN!

Bashscripting 104

An einem einfachen Beispiel mit „sleep" gezeigt:

```
[marius eve ] sleep 1d
[... keine Ausgabe mehr für einen Tag..]
```

Schauen wir uns so etwas mal im Prozessbaum an:

```
marius    12498  0.1  0.2 554436 44824 ?        Sl   09:35   0:01 /usr/libexec/gnome-terminal-server
marius    12541  0.0  0.0 116732  7444 pts/1    Ss   09:35   0:00  \_ bash
marius    13197  0.0  0.0 111676   692 pts/1    S+   10:00   0:00      \_ sleep 1d
```

Bashscripting 104

JETZT DIE VERSION MIT EINEM ABGESPLITTETEN PROZESS

```
[MARIUS EVE   ]  SLEEP 1D
[1] 13326
[MARIUS EVE   ]
```

JETZT WIEDER IM PROZESSBAUM, WO SICH BIS AUF DIE PID (PROZESSID) NICHTS GEÄNDERT HAT:

```
marius    12498  0.1  0.2 554436 44824 ?        Sl   09:35   0:02 /usr/libexec/gnome-terminal-server
marius    12541  0.0  0.0 116732  7444 pts/1    Ss+  09:35   0:00  \_ bash
marius    13326  0.0  0.0 111676   692 pts/1    S    10:04   0:00      \_ sleep 1d
```

Bashscripting 104

DER PROZESS LäUFT NOCH, **aber** ICH KANN WEITERARBEITEN!

Bashscripting 104

WAS PASSIERT JETZT ABER,

WENN ICH DIE BASH MIT DEM „SLEEP" BEENDE?

Bashscripting 104

DAS WAR DIE EINGABE BISLANG:

```
[MARIUS EVE ] SLEEP 1D
[1] 13326
[MARIUS EVE ] EXIT
[ ... UND BASH WIRD BEENDET, TERMINAL GEHT ZU ...]
```

JETZT WIEDER IM PROZESSBAUM:

```
marius    12530  0.0  0.1 587880 22556 ?        Sl   09:35   0:00 /usr/libexec/xdg-desktop-portal-gtk
marius    13326  0.0  0.0 111676   692 ?        S    10:04   0:00 sleep 1d
root      13366  0.0  0.0  12640  6788 ?        Ss   10:05   0:00 sshd: [accepted]
```

Bashscripting 104

DER SLEEP-PROZESS LäUFT NOCH,

ABER DIE BASH UND DAS TERMINAL SIND WEG.

Bashscripting 104

WAS IST DARAN JETZT SO SCHLIMM?

Bashscripting 104

Was ist daran jetzt so schlimm?

Ich habe jetzt einen frei laufenden Prozess im PC, der fast keinem anderen mehr Rechenschaft ablegen muß. Im Fall von „sleep" ist das jetzt nicht dramatisch, aber nehmen wir mal unseren Datenbank-Aufräumjob von vorhin als Basis.

Sagen wir mal „normal" wäre, wenn der Prozess für den Job *nicht länger als 2 Minuten braucht.* Braucht er länger, soll vielleicht der Datenbankadmin benachrichtigt werden, weil dann was „komisch" ist in der Datenbank. Könnte defekt sein, oder jemand hat sehr, sehr viele Daten in der Datenbank abgelegt.

Bashscripting 104

ABER WIE SOLL UNSER SCRIPT DAS JETZT NOCH MERKEN,

ES WURDE DOCH SCHON . . . BEENDET!

Bashscripting 104

Die Lösung lautet, warten bis es fertig ist und Zeit zählen:

```
#!/bin/bash

c=0
coproc dbclean ( mysql -p09djs22ss typo3datenbank -e „truncate t3_cache" )
while [ 1 ]; do

      rc=$(ps auxf| grep -c typo3datenbank)
      if [ $rc -lt 1 ]; then
            exit;   wenn der prozess vor 120 sekunden beendet wird, einfach direkt beenden. alles ok!
      fi
      sleep 1s
        c=$(( c+1))
      if [ $c -gt 120 ]; then
            ... mail dbadmin ...
            exit;   nach 120 sekunden, email an admin und dann script beenden, der dbjob kann weiterlaufen bis er durch ist!
      fi
done
  es ist nur ein beispiel!
```

Bashscripting 104

Die Folgen

Der Cronjob läuft jetzt maximal 2 Minuten (und ein paar Atosekunden) und beendet sich in jedem Fall selbst.

Der DB Admin bekommt seine Email und der DB Job kann weiterlaufen bis er durch ist.

Alle Gewinnen?

Bashscripting 104

FRAGEN!

Bashscripting 104

Anmerkungen

Es gibt natürlich noch andere Möglichkeiten Prozesse zu kontrollieren, z.B. in dem der Prozess ein PID File erzeugt, dessen Existenz in einer Schleife abgefragt wird.

```
while [ 1 ]; do if [ -f /var/run/prozessname.pid ]; then exit(); fi; sleep 1s; done;
```

Auch kann man mit dem Prozess kommunizieren, in dem stdin und stdout umgeleitet werden. *In unserem Beispiel war dies nicht nötig oder sinnvoll.*

Bashscripting 104

DANKE!