TEC WORKSHOP

Link: https://www.tecchannel.de/a/shell-scripting-ablaeufe-automatisieren,2032466

Linux-Workshop
Shell Scripting - Abläufe automatisieren

Datum: 17.09.2012
Autor(en): Thomas Steudten

Unter Linux lassen sich mit Scripts stets wiederkehrende Aufgaben erleichtern und automatisieren. In unserer Artikelserie über das Shell Scripting beginnen wir mit der grundlegenden Bedienung und den Möglichkeiten der Linux-Shells "Bourne" und dessen Nachfolger "Bash".

Das Betriebssystem **Linux**[1] zeigt seine Stärken für den Anwender auf der so genannten Kommandozeile. Natürlich sind mittlerweile grafische Oberflächen (Graphical User Interfaces GUI) weit verbreitet und haben auch ihre Daseinsberechtigung. Viele einfache Aufgaben sind damit auch problemlos per Maus zu erledigen. Stehen aber automatisierte und komplexere Abläufe an, so bietet es sich geradezu an, die Shell als Kommandointerpreter einzusetzen.

Besonders für System-Administratoren stellt das Shell Scripting schnell eine Arbeitserleichterung dar. So lässt sich beispielsweise einfach überprüfen, wo Dateien generiert oder geändert wurden. Prozesse können per Kommando angehalten werden, um sie später bei weniger Last auf dem System weiter auszuführen.

In insgesamt drei Artikeln des Shell Scriptings beschäftigen wir uns mit alltäglichen Machenschaften des Systemadministrators in Interaktion mit der Bourne-Shell ("sh") und dessen Nachfolger Bourne-Again-Shell ("bash"). In diesem Artikel beginnen wir mit der grundlegenden Funktionalität der Shell.

Artikelserie Shell-Scripting unter Linux

Teil 1: Shell Scripting - Abläufe automatisieren
Teil 2: **Shell Scripting im Admin-Alltag**[2]
Teil 3: **Shell Scripting - Tipps und Tricks für Admins**[3]

1. Grundlegenden Systemstatus abrufen

CTRL-X und CTRL-V präsentieren die Version und ein "echo $SHELL" den Pfad der Shell. Der Standard-Prompt wird über die Variable "PS1", die weiteren über "PS2" und "PS3" gesetzt. Ein "echo $USER $UID $EUID" zeigt uns den Usernamen und die (effektive) User-ID (UID) unter der wir angemeldet sind.

Mit "uname -a" ermitteln wir die Kernel-Version und die Architektur. Ist das /proc Dateisystem gemounted, so lassen sich mit "cat /proc/cpuinfo" Daten über die vorhandenen Prozessoren anzeigen. Welche letzten 10 Benutzer auf einem Multiuser-System angemeldet waren, zeigt uns "last -10" und die 3 letzten System-Reboots mit "last -3 reboot".

Traditionell steht der Runlevel S für Single-, 3 für Multiuser und 5 für Start des Grafiksystems. Diese Definition lockert sich je nach Distribution. "who -r" zeigt uns den aktuellen Runlevel und "uptime" die Zeit, seit dem das System "up", also eingeschaltet ist.

Der Befehle "mount", "df -h", und "du -sh" listet die aktiven Dateisysteme inklusive Type, Dimensionen und freien Platz in einer verständlichen Form ("-h") auf.

Ob unsere Shell eine interactive Shell ist, zeigt uns das Flag "i" in der Variablen "$-", nämlich "echo $- | grep -q i && echo interactive".

2. Shell-Skript

Der Grundaufbau eines Shell-Skripts sollte bei Meldungen den Namen den Skripts mit ausgeben und vor allem einen gültigen Exit-Wert:

```
> ME=$(basename $0); echo "$ME: Problem .."
```

Wie uns

```
> true ; echo $? ; false ; echo $?
```

verdeutlicht, steht der Wert "0" für eine erfolgreiche Ausführung, jeder andere Wert wäre nicht erfolgreich. Für die Shell steht "$? " für den letzten Exit-Wert eines Kommandos.

Wenn das erste Skript bei der Eingabe des Namens von der Shell nicht ausgeführt wird, kann dies mittels "bash myscript.sh" erreichen, oder man gibt dem Skript die für eine Ausführung nötigen Rechte per "chmod u+x myscript.sh". Als Superuser/ ROOT (UID=0) scheint die Shell das Script nocht oft nicht zu finden, obwohl dieses im aktuellen Verzeichnis vorliegt. Der Grund dafür ist, daß das aktuelle Verzeichnis nicht im Suchpfad der Shell vorhanden ist - für ROOT übrigens aus Sicherheitsgründen. Zwei Wege führen einfach zum Ziel:

```
# ./myscript.sh
```

oder:

```
# export PATH=.:$PATH; myscript.sh
```

Welche Zeilen im Script wie ausgeführt werden, ermöglicht der Aufruf der Shell mit dem Argument "-x", also "bash -x myscript.sh".

3. Pipe - Prozesse verknüpfen

Jeder gestartete Prozess hat im Minimum drei Datei-Handles (Filedescriptoren) offen:

1. Standard-In (stdout): Eingabe Handle ID 0
2. Standard-Out (stdin): Ausgabe Handle ID 1
3. Standard-Err (stderr): Fehlerausgabe Handle ID 2

Die Ausgabe des einen Prozesses kann mit der Eingabe eines anderen Prozesses verknüpft werden. Hier liegen gerade die Stärken von Linux. Der Shell teilt man dies mittels der so genannten Pipe "|" mit. Damit wird die Shell angewiesen, beide Prozesse auszuführen und die Dateihandles miteinander zu verbinden.

Beispielsweise zeigt die Folge "cat /etc/passwd | grep --color root" alle Zeilen in /etc/passwd, die die Zeichenfolge "root" enthalten. "cat" gibt alle Zeilen auf stdout aus, "grep" liest von stdin. Bei einem so einfachen Bespiel liese sich dies natürlich zu "grep -color root /etc/passwd" optimieren.

Eine ganze Kette von Filtern kann so aufgebaut werden:

cmd1 | cmd2 | cmd3 | .. | cmdn

Möchte man stderr und stdout jeweils in eine andere Datei umleiten, dann geht dies per "cmd 2> error.txt > out.txt". Eine kombinierte Version in nur eine Datei wäre "cmd > errout.txt 2>&1". Die Syntax dabei ist folgende:

1. Verknüpfe stdout von cmd mit der Datei "errout.txt"
2. Leite stderr dahin um, wohin stdout zeigt

4. Prozess-Steuerung über die Tastatur

Die Shell nimmt interaktiv ihre Befehle über die Tastatur vom Benutzer entgegen und führt diese nacheinander aus. Wir können aber der Shell auch mitteilen, dass diese in den Hintergrund geschickt werden soll und so die Tastatur wieder für andere Aufgaben verfügbar ist.

Ein "find / -type f -name date" würde bei Eingabe die aktuelle Shell beschäftigen bis das Kommando sich beendet hat, was unter Umständen sehr lange dauern kann. Also ab damit in den Hintergrund mit "find / -type f -name date &".

Das "&" - auch bekannt als "Ampersand" - teilt der Shell mit, die Folge im Hintergrund auszuführen. Dies können wir verifizieren und uns den Status der Hintergrundprozessgruppe anschauen mit "jobs" - dort sollte "Running" und "find" erscheinen. Die Job-ID steht in eckigen Klammern.

Möchte man den zuletzt in den Hintergrund geschickten Aufruf wieder in den Vordergrund holen, um diesen beispielsweise zu beenden, dann genügt ein "fg" oder "fg <jobid>", wobei "<jobid>" die ID kennzeichnet.

5. Abarbeitung stoppen und wiederaufnehmen

Nun läuft der Aufruf zwar im Vordergrund, aber möchten wir dies wirklich? Falls nicht, halten wir die Prozesskette mittels "CTRL-Z" einfach an. "jobs" zeigt uns ebenfalls nach Stunden den Status "Stopped" an und wir entscheiden uns, für eine Abarbeitung im Hintergrund, nachdem das System nicht mehr so unter Last steht: "bg" oder "bg <jobid>".

Leider schreibt uns die Ausgabe unser Terminal voll, also brechen wir die Ausführung mit "fg" und "CTRL-C" hier ab und senden die Fehlermeldungen ins Nirwana und die Ausgabe von "find" in eine Datei:

```
find / -type f -name date 2>/dev/null > /tmp/findout.txt
```

"2>" leitet nur stderr um, "1>" oder ">" die Standardausgabe in unsere Datei. Das ganze schicken wir in den Hintergrund.

Die Shell informiert uns, wenn die Abarbeitung beendet ist, sobald diese einen Prompt (PS1) ausgibt. Möchte man über die Beendigung sofort informiert werden, muss man der Shell dies über die Option "notify" mitteilen: "set -b" zum unmittelbaren Status oder "set +b" für Nachricht beim Prompt. "set -o" zeigt den Status der Optionen.

Stop und Resume sind nur zwei hilfreiche Tastatur-Eingaben um die Ausgabe im Terminal anzuhalten und wieder zu Starten. "CTRL-s" hält dazu die Ausgabe an und "CTRL-q" startet diese wieder. Die aktuelle Tastaturbelegung für die genannten Funktionen lassen sich über die Terminaleinstellungen "stty -a" abfragen und auch setzen.

Aus dem täglichen Leben kaum noch wegzudenken ist die History-Funktion der Shell über die Tasten Cursor-UP und -DOWN. Die ganze History seit dem Start der Shell lässt sich über das integrierte "history" Kommando ausgeben. Dort enthaltene Einträge kann man beispielsweise über "!34" wieder ausführen - 34 wäre hier der Eintrag mit der Nummer 34.

6. Datei-Konsistenz bei Backup / Restore überprüfen

Oft möchte man verifizieren, ob Dateien nach dem Backup und Restore noch konsistend sind, oder ob es viele Dateien mit identischem Inhalt, aber verschiedenen Dateinamen und/oder Datei-/Zeitstempel im Dateisystem gibt.

Am einfachsten generiert man einen Hash-Wert, beispielsweise mit dem fast überall verfügbaren "md5sum".

Die Ausgabe sieht dabei wie folgt aus: "<hashwert> /pfad/datei"

Checksumme über Dateipfad bilden:

```
find /bin /sbin /usr/bin /usr/sbin -type f -exec md5sum {} \; > /tmp/MD5SUM_bin.txt
```

Nach dem Backup und Restore lassen sich dann diese Checksummen erneut bilden und mit den gespeicherten vergleichen:

```
md5sum -c /tmp/MD5SUM_bin.txt | grep -v OK
```

7. Dubletten und aktuelle Dateien aufspüren

Um mehrfach vorhandene Dateien (Dubletten) aufzuspüren, ist die Hash-Datei zu sortieren und mehrfach vorhandene und gleiche Hash-Werte auszugeben:

```
sort -k1,1 /tmp/MD5SUM_bin.txt | uniq -w 33 -D
```

Nun muss man sich entscheiden, welche Datei in welchem Pfad man behalten und welche löschen möchte. Möchte man ermitteln, welche Dateien am aktuellen Tag (seit Mitternacht) angelegt wurden, ohne auf die "-newer"-Syntax mit einer Vergleichsdatei zurückzugreifen, so bietet die Option "-d" von "date" die passende Lösung. "date -d'' liefert den Zeitstempel für Mitternacht zurück, das heißt Stunde, Minute und Sekunde ist jeweils Null.

Unterstützt der Befehl "find" die "-newerXY" Syntax, dann sieht der Aufruf wie folgt aus:

```
find /var/spool -type f -newerct
```

Last but not least: "dmesg" gibt den Inhalt des Kernel-Ring-Buffers aus und das sind meist die Meldungen des Kernels und der Treiber, die auch über den "syslogd" nach "https://www.computerwoche.de/var/log/messages" geschrieben werden. Unmittelbar nach dem Booten, lassen sich dort nützliche Informationen entnehmen. (cvi)

Links im Artikel:

[1] **https://www.tecchannel.de/server/linux/**

[2] **https://www.tecchannel.de/server/linux/2033265/linux_workshop_shell_scripting_fuer_den_admin_alltag/**

[3] **https://www.tecchannel.de/server/linux/2033904/linux_shell_scripting_tipps_und_tricks_fuer_admins/**
