TEC WORKSHOP

Link: https://www.tecchannel.de/a/shell-scripting-im-admin-alltag,2033265

Linux-Workshop
Shell Scripting im Admin-Alltag

Datum: 17.08.2011
Autor(en): Thomas Steudten

Bei Linux erfolgt die Interaktion mit dem System oft über die Shell via csh, ksh, sh, pdsh oder bash. Wer die richtigen Kommandos und die entsprechende Syntax allerdings nicht beherrscht, verzweifelt schnell. Wir zeigen Ihnen deshalb typische Bash-Beispiele aus der Praxis des Admins.

Für System-Administratoren stellt das Shell Scripting unter Linux schnell eine Arbeitserleichterung dar. Ist man mit der grundlegenden Bedienung und den Gepflogenheiten der Bash vertraut, so lassen sich Abläufe bequem automatisieren. Auch lässt sich auf der Kommandozeile viel mehr erledigen, als mit den üblichen graphischen Admin-Tools möglich ist.

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").

Im ersten Linux-Workshop **Shell Scripting - Abläufe automatisieren**[1] geht es um die grundlegende Funktionalität der Shell. In diesem Artikel befassen wir uns mit dem Admin-Alltag auf der Shell und zeigen typische Scripting-Beispiele.

1. Datei-Handling

Hin und wieder erhalten wir nachfolgende Meldung, wenn es darum geht, eine hohe Zahl an Dateien zu Durchsuchen oder Logdateien zu Löschen:

> ls *.gz

-bash: /bin/ls: Argument list too long

Die einem Prozess übergebene Zahl an Argumenten wurde überschritten. Wenn die Option "noglob" in der Shell deaktiviert ist (default; set +-f; Anzeige: set -o), dann wird der Ausdruck "*.gz" so expandiert, dass alle Dateien mit der Endung ".gz" im aktuellen Verzeichnis gesucht, und dem Befehl "ls" als Argumentenliste übergeben werden. Effektiv führt die Shell daher den Aufruf "ls file1.gz file2.gz .. fileN.gz" aus, wenn diese Dateien vorhanden sind.

Würden wir das so genannten Globbing, das heißt die Expansion von "*?" durch die Shell, mittels "set -f" deaktivieren, dann hat "*" für die Shell keine besondere Bedeutung mehr.

> set -f

> zgrep test *.gz

gzip: *.gz: No such file or directory

> touch /tmp/*.gz /tmp/a.gz

> ls -l /tmp/*.gz

-rw-rw-r-- 1 user grp 0 Dec 7 10:55 /tmp/*.gz

Globbing wieder aktiv:

> set +f

> ls -l /tmp/*.gz

-rw-rw-r-- 1 user grp 0 Dec 7 10:57 /tmp/a.gz

-rw-rw-r-- 1 user grp 0 Dec 7 10:55 /tmp/*.gz

Unserem Befehl "ls/ zgrep" müssen wir daher die Dateien häppchenweise servieren.

Möglichkeiten:

ls [a-h]*.gz ; zgrep test [i-n]*.gz; ls [o-z].gz

find . -type f -name '*.gz' -exec zgrep test {} \;

find . -type f -name '*.gz' | xargs -n 200 zgrep test

Unter der Annahme, dass die jeweilige Expansion von "[a-h]*.gz", "[i-n]*.gz" und "[o-z]*.gz" die maximale Anzahl an Argumenten nicht überschreitet, wird jeweils nur ein Teil der Dateien dem Aufruf bei Option 1. übergeben.

Bei Option 2. expandiert der Befehl "find" den Ausdruck ‚*.gz' selbst, daher ist es ein Unterschied, ob wir "*.gz" oder "*.gz" übergeben. Im ersten Fall wird auch hier wieder die Shell aktiv und expandiert diesen Ausdruck, was wir nicht möchten. Die Option 1. automatisiert ergibt den Aufruf bei Möglichkeit 3. Hier liest "xargs" maximal 200 Dateinamen über stdin ein, führt mit dieser Liste das nachfolgende Kommando aus. Dies wiederholt sich, bis alle Namen verarbeitet wurden.

Wenn die Dateinamen aber Leerzeichen und/oder Zeilenumbruch enthalten, dann empfiehlt sich im Fall 3. die folgende Syntax, die den Wert 0 als Dateiseparator, statt Leerzeichen (ASCII-32) verwendet:

> find . -type f -name '*.gz' -print0 | xargs -0n 200 zgrep test

Ab Kernel Version 2.6.23 sollte diese Meldung nicht mehr sobald erscheinen, denn der Kernel wurde so angepasst, dass die Anzahl der Argumente nun dynamisch angepasst wird (zirka 25% von ulimit -s; der Prozessstack-Grösse).

Kernel 2.6.18:

> ls *.gz

-bash: /bin/ls: Argument list too long

> find . -name '*.gz' |wc -l

6922

Kernel 2.6.35:

> find . -name 'test*' | wc -l

150000

2. Achtung bei Groß- und Kleinschreibung

Dem aufmerksamen Leser wird aufgefallen sein, dass "[a-h]*.gz" auch Dateien mit großem Anfangsbuchstaben erfasst, während "a*.gz" und "A*.gz" unterschiedlche Ergebnisse liefern:

> touch Haus haus

> ls h*

haus

> ls H*

Haus

> ls [a-z]*

haus Haus

> ls [A-Z]*

haus Haus

Wird die Shell-Option "nocaseglob" gesetzt, dann ist das Globbing case-insensitive, das heißt unabhängig von Groß- und Kleinschreibung:

> shopt | grep --color glob

dotglob off

extglob on

failglob off

globstar off

nocaseglob off

nullglob off

> shopt -s nocaseglob; shopt | grep --color nocaseglob

nocaseglob on

> ls h*; ls H*

haus Haus

haus Haus

Der Vorteil von "find" ist die Möglichkeit der Suchbegrenzung beispielsweise auf Dateien, die seit Mitternacht angelegt wurden oder explizit nur Dateien, keine Links oder Verzeichnisse. Denn "ls [a-h]*.gz" würde auch Verzeichnisse beziehungsweise die Dateien darin erfassen.

3. Ausgaben-Verarbeitung

Die Ausgaben von Befehlen (stdout) lassen sich einfach weiterverarbeiten, indem diese einer Shell-Variablen zugewiesen werden (Command Substitution). Nachfolgende Zeilenumbrüche werden dabei entfernt.

Dazu bietet die Bash den Klassiker der Hochkommata-nach-Links (Backquote) md=`date` Version und md=$(date).

Mittels "set $md" können wir uns die durch Leerzeichen getrennte Werte in den Positionsparametern $1, $2, ..$n zugänglich machen.

> md=`date`; set $md; echo $4

14:30:28

> date ; uname; whoami

Wed Dec 8 11:31:53 CET 2010

Linux

thomas

> a=$(date ; uname; whoami); echo $a

Wed Dec 8 11:32:05 CET 2010 Linux thomas

Bei einer Verschachtelung sind die inneren Backquotes mit dem Bash- Escape-Zeichen '\' zu maskieren:

> set -x

> echo `set -- \`date\`; echo $4`

+++ date

++ set -- Wed Dec 8 11:40:31 CET 2010

++ echo 11:40:31

+ echo 11:40:31

11:40:31

Eine optimierte Version von "$(cat file)", bei der vorgängig noch der Befehl "cat" aufgeführt wird, bietet die bash mit "$(< file)".

4. Pfad-Verarbeitung

Oft möchte man nur den Datei- oder Verzeichnisnamen ohne den vollen Pfad nutzen. Nachfolgend sind einige Beispiele gezeigt:

> basename /bin/date

date

> dirname /bin/date

/bin

ME=$(basename $0)

DIR=$(cd $(dirname $0); pwd)

SETUP=$(cd $DIR/../etc; pwd)/$ME.conf

echo "$ME: running from location $DIR.."

echo "$ME: loading setup from $SETUP.."

. $SETUP

echo $LOGLVL

Das Skript "path.sh" lädt das Setting abhängig von dessen Verzeichnisablage und Dateinamen immer eine Ebene höher in etc/<scriptname>.conf:

> pwd

> bash tmp/path.sh

path.sh: running from location /home/thomas/tmp..

path.sh: loading setup from /home/thomas/etc/path.sh.conf..

2

> cd tmp; ./path.sh

path.sh: running from location /home/thomas/tmp..

path.sh: loading setup from /home/thomas/etc/path.sh.conf..

2

> mv path.sh newpath; ./newpath

newpath: running from location /home/thomas/tmp..

newpath: loading setup from /home/thomas/etc/newpath.conf..

./newpath: line 10: /home/thomas/etc/newpath.conf: No such file or directory

5. Verzeichnis-Stack

Einen schnellen Wechsel zu einem anderen Verzeichnis und zurück lässt sich mit "pushd" und "popd" praktizieren:

> pwd

/home/thomas/tmp

> pushd /var/spool/cron

/var/spool/cron ~/tmp

> dirs

/var/spool/cron ~/tmp

> pushd /bin

/bin /var/spool/cron ~/tmp

> popd; popd

~/tmp

> popd

bash: popd: directory stack empty

6. Alias

Wenn man eine längere Befehlskette öfters eintippen möchte, lässt sich dies mit dem alias-Feature der Shell vereinfachen. Ein alias ist aber nur interaktiv gültig, das heißt in einem Skript ist die Funktion zu nutzen.

Folgendes Kommando zeigt die aktuell definierten Aliase:

> alias

Dieses Kommande definiert einen alias "myf":

> alias myf="find . -type f -size +1M -ls 2>/dev/null"

Aufruf des Alias:

> myf

794679 1784 -rw------- 1 root root 1822506 Oct 12 08:46 ./dist-upgrade/apt-term.log

Löscht den alias "myf" wieder:

> unalias myf

Eine erneute alias-Definition mit dem gleichen Namen überschreibt den ursprünglichen Inhalt. Findet die Shell auf der Promptzeile eine Übereinstimmung mit dem Namen einer Shell-Funktion und dem Namen eines Alias, so wird die Alias-Expansion ausgeführt. Im Unterschied zur Funktion, bei der wir Argumente beliebig platzieren können, ersetzt die Shell lediglich den Alias durch den dazugehörigen Text:

> type tom1

bash: type: tom1: not found

```
> tom1() { echo tom-func; md5sum $1; date; ls -l $2; }

> type tom1
tom1 is a function
tom1 ()
{
echo tom-func;
md5sum $1;
date;
ls -l $2
}

> tom1 /bin/date /etc/passwd
tom-func
fe7ae39c0adc727bad660350d24f5d68 /bin/date
Wed Dec 8 08:42:05 CET 2010
-rw-r--r-- 1 root root 2173 2010-10-01 13:52 /etc/passwd

> alias tom1="echo tom-alias; whoami"
> type tom1
tom1 is aliased to `echo tom-alias; whoami'
> tom1
tom-alias
thomas
```

> unalias tom1

> type tom1

tom1 is a function

[..]

Die Ersetzung des Alias auf der Promptzeile durch den dazugehörigen Text führt dazu, dass dieser Text ebenfalls in den dort neu definierten Funktionen und Aliase auftaucht, selbst wenn der Alias nicht mehr existiert:

> type ls

ls is hashed (/bin/ls)

> alias ls="ls --color"

> type ls

ls is aliased to `ls --color'

> type tom1

tom1 is a function

tom1 ()

{

echo tom-func;

md5sum $1;

date;

ls -l $2

}

> tom1() { echo tom-func; md5sum $1; date; ls -l $2; }

> unalias ls

> type tom1

tom1 is a function

```
tom1 ()

{

echo tom-func;

md5sum $1;

date;

ls --color -l $2 <= "--color": stammt aus der Alias Def. von ls

}
```

7. Rechnen in und mit der Bash

Bekannt sein dürfte der Klassiker "expr", der über viele Plattformen Kompatibilität bringt:

```
count=10


while [ $count -gt 0 ]; do

echo $count

count=$(expr $count - 1)

done
```

Alternativen aus der Bash:

- let count=$count-1
- count=$((count-1))
- count=$[count-1]

8. Brace-Expansion

Unter Brace-Expansion versteht die Bash das Ersetzen von Ausdrücken bei Wörtern, die mit der geschweiften Klammer '{}' beginnen oder enden. So wird beispielsweise der Ausdruck "echo tom{1,2}" zu tom1 tom2 expandiert und "echo /bin/d{a,d}*" zu /bin/dash /bin/date /bin/dd - Dateien im /bin Verzeichnis. Diese Syntax ist jedoch zur Bourne-Shell nicht kompatibel:

```
> sh -c "echo tom{1,2}"

tom{1,2}
```

> bash -c "echo tom{1,2}"

tom1 tom2

9. True or False? Testing...

Bash hat die Funktionalität von "test" beziehungsweise "[" als Builtin integriert.

> type test [

test is a shell builtin

[is a shell builtin

Elementare Tests auf Dateisystemebene helfen dem Admin, zu testen, ob eine Datei existiert, ob diese leer ist, ob es sich um eine Datei, ein Verzeichnis, einen Softlink, einen Socket oder eine FIFO handelt. Sind zwei Ausdrücke identisch [string1 = string2] oder ist eine Integervariable größer 5 "test $count -gt 5"?

Möchte man testen, ob stdin oder stdout an ein Terminal gebunden ist, das heißt die Eingabe erfolgt nicht über eine Pipe oder Dateiumlenkung, dann hilft die Syntax "test -t 0" bzw. "test -t 1". Damit lassen sich dann beispielsweise Statusmeldungen unterdrücken, wenn über eine Pipe gelesen wird. (cvi)

Links im Artikel:

[1] **https://www.tecchannel.de/server/linux/2032466/workshop_shell_scripting_fuer_linux/**
