Die Shell - Details

Die Shell, die Kommandozeile und ein Teil vom ganzen Rest

Dirk Geschke

Linux User Group Erding

26. November 2008

Gliederung

Aufgaben (bash)

- Schnittstelle zwischen Kernel und Anwender
- Jobkontrolle: Starten, stoppen, beenden, Vordergrund, Hintergrund, Signale
- programmierbar: Shellskripte, Funktionen, Aliase
- eingebaute Kommandos, z.B. `cd, echo, fg, bg, alias, pwd, kill, ...`
- Umgebungsvariablen, z.B. `HOME, PATH, LANG, ...` und Vererbung
- Features: Vervollständigung, History, editierbare Kommandozeile

Umgebungsvariablen

- Werden in der shell einfach gesetzt, z.B.: `PS1='\h:\w> '`
- Vererbung an Subshell mittels export, z.B.: `export PS1`
- Ausgabe mit printenv oder set
- Ausgabe mit echo: `echo $PATH`
- erweiterbar, z.B.: `PATH=$PATH:/opt/bin`

Vordefinierte Variablen

$?	Rückgabewert des letzten Kommandos
$!	PID des zuletzt gestarteten Hintergrundprozesses
$$	aktuelle PID der Shell
$0	Name des ausgeführten Shell-Scripts
$#	Anzahl der übergebenen Parameter
$1 - $9	Parameter 1 bis 9
$* oder $@	Alle übergebenen Parameter
~	HOME-Verzeichnis

Kommandoaufrufe

- Einfach am Prompt eingeben: `ls`
- kurze Optionen mit einfachem Minuszeichen anhängen, z.B.: `ls -l -d` oder auch `ls -ld`
- lange Optionen mit zwei Minuszeichen, z.B.: `ls --help`
- kurze Hilfe oft mittels –h oder `--help`
- Suchpfad relevant: $PATH
- Pfadangabe verwenden:
  - absolut: `/bin/ls`
  - relativ: `./ls`
- starten im Hintergrund: & anhängen, z.B.: `xpdf shell.pdf &`

Jobkontrolle

- Jedes Kommando erhält eine Prozess-ID: PID
- Kontrolle der Jobs via Signale, diese können via kill gesendet werden: `kill -{signal} {pid}`, z.B.:
  `kill -INT 730`
- Signale können auch via Tastenkürzel gesendet werden:

SIGINT	STRG-C, Abbruch des Kommandos
SIGABRT	STRG-\, Abbruch mit core-Datei
SIGSTOP	STRG-Z, Anhalten eines Prozesses
trap	Shellkommando zum Abfangen von Signalen

- Prozesse in den Hintergrund mit `bg`, in den Vordergrund mit `fg`
- Auflisten von laufenden Prozessen: `jobs`
- Referenzierung mit `%` und Jobnummer, z.B.: `fg %1`

Einige Hilfreiche Tastenkürzel

Cursortasten	hoch und runter: Navigation durch Historie
Cursortasten	links und rechts: Navigation durch Zeile
STRG-A	Anfang der Kommanozeile anspringen
STRG-E	Ende der Kommanozeile anspringen
STRG-L	Löschen des Bildschirms
STRG-K	Löschen des Rests der Zeile
STRG-R	inkrementelle Rückwärtssuche in Historie
STRG-S	Einfrieren der Ausgabe, eigentlich Vorwärtssuche
STRG-Q	Reaktivieren der Ausgabe
ESC-.	Letztes Argument der vorherigen Zeile
TAB-Taste	Pfad-, Kommando-, Dateiergänzung

Aliase und Funktionen

- Aliase können mit alias gesetzt werden, z.B.:
  `alias more=less`
  Ohne Argument erfolgt die Auflistung definierter aliase
- Aliase können mit unalias gelöscht werden, z.B.:
  `unalias ls`
- Funktionen können auch leicht definiert werden, z.B.:
  `listDG() { ls $1; }`
  Aufruf mit `listDG s*` oder `listDG` `` `s*` ``
  Unterschied: Expansion in der Shell
- Löschen erfolgt mit unset, z.B.: `unset listDG`

Ausgabeumlenkung

- Ausgabe kann in Datei umgelenkt werden mittels >, z.B.: `ls -l > Listing`
- Eingabe kann aus Datei erfolgen, z.B.: `cat < Listing`
- drei Arten von Standardkanälen:

STDIN	Standardeingabekanal, FD 0
STDOUT	Standardausgabekanal, FD 1
STDERR	Standardfehlerkanal, FD 2

- auf STDERR werden typischerweise Fehlermeldungen und Hilfetexte ausgegeben
- Trennung der Kanäle sinnvoll kann aber aufgehoben werden:

2 >	Umlenkung von STDERR
2 > &1	Umlenkung von STDERR nach STDOUT
1 > &2	Umlenkung von STDOUT nach STDERR

- Anhängen an Datei mit >> möglich

Die Pipe

- Ein Programm kann in eine Datei schreiben, das nächste daraus lesen, z.B.:

```
ls -l > Listing; wc -l < Listing
```

- Das geht aber auch eleganter mit einer pipe:

```
ls -l | wc -l
```

- Viele Programme können wahlweise direkt aus einer Datei lesen oder von STDIN
- Umlenkung von STDERR in eine pipe geht auch:

```
ls -y 2>&1 > Listing | less
```

- gleichzeitiges Schreiben in Datei und STDOUT ist mit `tee` möglich:

```
ls -l | tee > Listing | wc -l
```

Substitutionen

Ausdruck	Bedeutung
?	genau ein Zeichen
*	beliebig viele Zeichen
[a-z]	ein Zeichen aus dem Bereich von a bis z
[!a-z]	kein Zeichen aus dem Bereich von a bis z
[^a-z]	kein Zeichen aus dem Bereich von a bis z
~	HOME-Verzeichnis
.	aktuelles Verzeichnis
..	übergeordnetes Verzeichnis
‘ Befehl ‘	Ausführung des Befehls in einer Subshell
$(Befehl)	Ausführung des Befehls in einer Subshell

Pattern-Matching

Ausdruck	Bedeutung
${*var#muster*}	wenn *muster* am Anfang gefunden wird, entferne es und gib den Rest aus
${*var##muster*}	wie oben aber längster Treffer
${*var%muster*}	wie oben aber vom Ende, kürzester Treffer
${*var%%muster*}	wie oben aber längster Treffer
${*var/muster/ersetzung*}	längste erste Übereinstimmung von *muster* wird ersetzt
${*var//muster/ersetzung*}	wie oben, jedoch alle *muster* werden ersetzt

Pattern-Matching: Beispiele

Ausdruck	Ergebnis
$*var*	/home/geschke/LG/shell.2.pdf
${*var*#*/*/*}	geschke/LG/shell.2.pdf
${*var*#*#/*/*}	shell.2.pdf
${*var*%*.**}	/home/geschke/LG/shell.2
${*var*%%*.**}	/home/geschke/LG/shell
${*var*/*./:*}	/home/geschke/LG/shell:2.pdf
${*var*//*./:*}	/home/geschke/LG/shell:2:pdf

String-Operationen

Ausdruck	Bedeutung
${*var:-wert*}	wenn *var* definiert gib es aus, sonst gib *wert* aus
${*var:=wert*}	wenn *var* definiert gib es aus, sonst setze es auf *wert* und gib es aus
${*var:+wert*}	wenn *var* definiert gib *wert* aus
${*var:?Text*}	wenn *var* definiert gib es aus, sonst breche ab und gib *wert* aus
${*var:offset*}	Zeichenstring ab *offset*
${*var:offset:length*}	Zeichenstring ab *offset* der Länge *length*

Programmierung

- Viele eingebaute Kommandos aber auch externe können aufgerufen werden
- Funktionen liefern Rückgabewert:

  0 true, Ausführung war erfolgreich

  $\neq 0$ false, ein Fehler ist aufgetreten
- Es gibt ein `test` Kommando, auch via [und] darstellbar:
  - Datei existiert: test
  - Datei existiert und ist ausführbar: test -x
  - Datei- und Stringvergleiche sind auch möglich
  - Beispiel für Anwendung von `test`:

```
qfix:~/LG> test shell.tex; echo $?
0
qfix:~/LG> test -x shell.tex; echo $?
1
qfix:~/LG> [ -x shell.tex ]; echo $?
1
```

Programmierung

- for-Schleifen, z.B.:

```
for i in *; do echo $i; done
```

- if-Abfragen, z.B.:

```
if [ -x /bin/bash ]
then
     echo Bash existiert und ist ausführbar
else
     echo Bash nicht existent oder ausführbar
fi
```

- Ergebnisse können direkt ausgewertet werden:

```
if /bin/true; then echo Ergebnis ist positiv; fi
```

Programmierung

- case–Verzweigungen:

```
case $i in
     hallo* ) echo Variable beginnt mit hallo;;
     *) echo kein hallo ;;
esac
```

- while–Schleife

```
while read i; do echo $i; done < Listing
```

- until–Schleife

```
i=first; until [ -z "$i"]
do read i; echo $i; done < Listing
```

Kommandoverknüpfung

Kommando1; Kommando2 erst Kommando1, dann Kommando2 ausführen

(Kommando1; Kommando2) das gleiche wie oben aber in der gleichen Shell

Kommando1 && Kommando2 Kommando2 wird nur bei Erfolg von Kommando1 ausgeführt

Kommando1 || Kommando2 Kommando2 wird nur bei Fehler von Kommando1 ausgeführt

Kommando1 & Kommando2 Kommando1 wird im Hintergrund gestartet, Kommando2 im Vordergrund

Besonderheiten

- Kommentarzeichen ist #, erste Zeile `#! Kommando`: Auswahl des Interpreters
- Starten einer Shell-Scriptes in der akuellen Datei mittels `. Datei` oder `source Datei`
- Lage des Kommandos: `which Kommando`, z.B.:

```
qfix:~> which ls
/bin/ls
```

- Analog Shell-Builtin `type` aber mit Unterschieden:

```
qfix:~> which pwd
/bin/pwd
qfix:~> type pwd
pwd is a shell builtin
```

Hilfe finden

- Manualpages: `man Kommando`
  - `man -k Stichwort` oder `apropos Stichwort`: liefert liste an Hilfeseiten zum Stichwort
  - `wahtis Kommando` liefert kurze Beschreibung der Manual-Page
- `info Kommando` liefert Info-Pages zum Kommando, oft sind das aber die Manualpages, GNU-Projekt verwendet diese
- doc-Verzeichnis: `/usr/share/doc/{Paket}/`
- FAQs: Frequently Asked Questions, Sammlung häufig gestellter Freagen
- HOWTOs: Beschreibung von Einrichtungen/Konfigurationen, basieren häufig auf FAQs

Dateiverwaltung

cd	Wechseln von Verzeichnissen (*change directory*)
cat	Ausgeben von Dateiinhalten (*con-cat-enate*)
ls	Inhaltsverzeichnis auflisten (*list*)
cp	Kopieren von Dateien (*copy*)
mkdir	Verzeichnis erstellen (*make directory*)
rmdir	Verzeichnis löschen, es muss leer sein (*remove directory*)
mv	Dateien verschieben oder umbenennen (*move*)
rm	Löschen von Dateien (*remove*)
file	gibt an um welche Art von Datei es sich handelt

Dateiensuche

find	Durchsuchen aller Unterverzeichnisse nach Muster
locate	Lokalisieren von Dateien anhand einer Datenbank
updatedb	Aktualisieren der *locate*–Datenbank
whereis	Finden von ausführbaren Programmen, den Sourcen und der Manual–Page
which	Auflisten des Pfades zu einem ausführbaren Programm
type	Analog zu `which`, sucht aber auch Funktionen, Aliase und Builtins (Shell–Builtin)

Bearbeitung von Textdateien

cut	einzelne Spalten aus Zeilen extrahieren
diff	Unterschiede zwischen Dateien finden
grep	nach regulären Ausdrücken in einer Datei suchen
head	die ersten Zeilen einer Datei ausgeben
tail	die letzten Zeilen einer Datei ausgeben
more	Seitenweise Ansicht von Dateien (*pager*)
less	komfortableres `more`, *less is more*
recode	Konvertierung von Zeichensätzen
sort	Sortieren, nummerisch oder alphabetisch
sed	Stream–Editor
split	Datei in Blöcke bestimmter Größe zerlegen
strings	Zeichenketten aus einer Binärdatei extrahieren
tr	ersetzen von Zeichen, z.B. von Gross- zu Kleinschreibung
uniq	Entfernen von doppelten Einträgen

Komprimieren und Archivieren von Dateien

bzip2/bunzip2	stärker als `gzip`
cpio	kopieren von Dateien zwischen unterschiedlich Devices
compress/uncompress	Lempel–Ziv–Kompimierung
gzip/gunzip	klassisches Standard–Packverfahren unter Unix
tar	erstellen von Archiv–Dateien oder Schreiben auf Bandlaufwerk
zip/unzip	Windows–kompatible Zip–Archive
zipinfo	Auflisten des Inhaltes eines ZIP–Archivs

Prorgramm und Prozessverwaltung

ps	Auflisten der laufenden Prozesse
pstree	Auflisten der laufenden Prozesse in einen Hierachie–Baum
top	Auflistung der Prozesse die am meisten Ressourcen verbrauchen
bg/fg	Verschieben in den Hintergrund/Vordergrund
kill	Senden von Signalen an Prozesse (PID)
killall	Senden von Signalen an Prozesse (via Namen)
nice/renice	Ändern der Priorität eines Prozesses
nohup	Abkoppeln eines Prozesses von der laufenden Shell
sudo	Prozess als Superuser ausführen

Benutzerverwaltung

id	Auflisten der UID und GIDs
chsh	Ändern der Login–Shell
groups	Listet alle Gruppen in denen man ist auf
newgrp	ändert die aktive Gruppe eines Users
passwd	ändern des Passwortes
adduser/deluser	Hinzufügen oder Löschen eines Users
useradd/userdel/usermod	Hinzufügen, Löschen oder Ändern eines Benutzers

Administration von Dateisystemen

du	listet wieviel Platz ein Verzeichnis belegt
dd	kopiert *low-level* zwischen zwei Devices
df	Auflistung der Auslastung von Partitionen
fdisk	Partitionierung von Festplatten
mkfs	Erzeugen eines Dateisystems
mount	Einbinden von Laufwerken
sync	erzwingt, dass alle Buffer geschrieben werden
mkswap/swapon/swapoff	einrichten, aktivieren, deaktivieren von Swap-Speicher

Kommandos für das Netzwerk

ifconfig	Administration von Interfaces
ip	neue Variante der Netzwerkadministartion
route	Anzeigen oder Ändern von Routen
ifup/ifdown	gezieltes Steuern von Netzwerkinterfaces
ethtool	Konfiguration einzelner Netzwerkkarte
netstat	Anzeigen von Verbindungszuständen

Weitere interessante Kommandos

dmesg	Ausgabe von Kernelmeldungen
date	Datum und Uhrzeit ausgeben
free	Speicherbelegung ausgeben
uname	Ausgabe des Betriebsystems und Kernelversion
xargs	Standardeingabe wird zeilenweise an anderes Kommando weitergeleitet
alien	konvertiert verschiedene Paketformate
lspci	Auflisten der PCI–Geräte
lsusb	Auflisten der gefundenen USB–Geräte

Starten des Kernels

1. Kernel wird in den Speicher geladen und entpackt sich selber
2. Initialisierung der CPU und des Hauptspeichers
3. Erkennung und Initialisierung der restlichen Hardware
4. optionales mounten von initrd, einer Ramdisk
5. Mounten der root-Partition (/)
6. Starten von init

Der Init-Prozess

- init wertet die Datei /etc/inittab aus und arbeitet sie ab
- Es gibt mehrere runlevel: 0 bis 6

0	halt, System wird angehalten
1	Single-User, nur ein Benutzer kann sich anmelden, kein Netzwerk
2-5	Multi-User, mit/ohne Netzwerk, mit/ohne X11
6	reboot, System wird neu gestartet

- Default-Runlevel bei Debian: 2

```
# The default runlevel.
id:2:initdefault:
```

Syntax der inittab

- Die Syntax der Datei ist id:runlevels:action:process

id	eindeutige ID aus bis zu 4 Zeichen, gewöhnlich 2
runlevels	listet die *runlevels* auf für welche die *action* ausgeführt werden soll
action	gibt an welche Aktion durchgeführt werden soll
process	welcher Prozess soll ausgeführt werden.

Das action–Feld, Auswahl

initdefault	gibt den runlevel an der beim Booten gestartet werden soll
sysinit	wird beim booten ausgeführt
wait	Der Prozess wird einmal gestartet wenn der runlevel betreten wird, danach wird auf das Ende des Prozesses gewartet
respawn	der gestartete Prozess wird bei Beendigung erneut gestartet, dies betrifft hauptsächlich die virtuellen Konsolen
ctrlaltdel	was soll bei CTRL-ALT-DEL passieren?

Kurzversion der inittab

```
id:2:initdefault:
si::sysinit:/etc/init.d/rcS
~~:S:wait:/sbin/sulogin
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
...
```

Abarbeitung von /etc/init.d/rc

- Zuerst werden die K-Prozesse gestoppt:

```
for i in /etc/rc$runlevel.d/K[0-9][0-9]*
do
     $i stop
done
```

- Danach die Startscripte aktiviert:

```
for i in /etc/rc$runlevel.d/S[0-9][0-9]*
do
     $i start
done
```

- Die Start-/Stopskripte sind typischerweise symbolische Links auf die Skritpe in `/etc/init.d/`

Skelett eines Startskriptes

Abgesehen von vielen Checks und Laden diverser Einstellungen aus anderen Dateien besteht ein Start/Stop–Skript aus diesem Fragment:

```
#!/bin/sh
case $1 in
  start)
    /usr/bin/Kommando && echo Kommando started
    ;;

  stop)
    killall Kommando && echo Kommando stopped
    ;;
  *)
    echo Usage: $0 {start | stop }
    exit 1
;;
esac
```