Linux – Prinzipien und Programmierung

Dr. Klaus Höppner

Hochschule Darmstadt – Wintersemester 2011/2012

Dateisystem-Shellbefehle

bash

Wechseln von Verzeichnissen

`cd` Wechsel in das angegebene Verzeichnis (falls keins angegeben: `$HOME`). Hierbei kann in der bash das Homeverzeichnis als `~` abgekürzt werden, so bezeichnet z. B. `~/Documents` ein Unterverzeichnis `Documents` im Home-Verzeichnis.

`pwd` Ausgabe des aktuellen Verzeichnisses.

`pushd, popd` Verzeichniswechsel mit „Merken“ der Verzeichnisse auf einem Verzeichnis-Stack (mit `pushd` wechseln und neuen Stackeintrag anlegen, mit `popd` zurück und letzten Stackeintrag entfernen).

Finden

grep Suche nach Zeichenketten innerhalb einer oder mehrerer Dateien.

find Rekursive Suche nach Dateien anhand des Namens, Datums etc.

locate Globale Suche nach einer Datei anhand ihres Namens (innerhalb einer Datenbank-Datei, die regelmäßig per `updatedb` aktualisiert werden sollte).

which Angabe des Pfades zu einem ausführbaren Programm, das innerhalb von `$PATH` liegt.

Vorteile der bash als Standard-Shell

- Großer Funktionsumfang
- Command Completion mit der TAB-Taste
- History-Funktionen:
  - Blättern
  - Suchen
  - Wiederholen alter Kommandos

Ausführen von Kommandos

In der Bash können folgende Befehle eingegeben werden:

- Interner Bash-Befehl (z. B. `pusdh`),
- Programm, das innerhalb der in `$PATH` enthaltenen Verzeichnisse gefunden wird,
- Programm mit (absoluter oder relativer) Pfadangabe.

Kommandos können kombiniert werden:

`cmd1; cmd2` cmd1 und cmd2 werden hintereinander ausgeführt,

`cmd1 && cmd2` cmd2 wird dann ausgeführt, wenn cmd1 erfolgreich war,

`cmd1 || cmd2` cmd2 wird dann ausgeführt, wenn *cmd1* nicht erfolgreich war.

Beispiele

`test -d ~/foo && rm -rf ~/foo`
Falls das Verzeichnis `~/foo` existiert, wird es (rekursiv) gelöscht.

`test -d ~/foo || mkdir ~/foo`
Das Verzeichnis `~/foo` wird angelegt, falls es noch nicht existiert.

Alternative zu `test -d ~/foo`:
`[ -d ~/foo ]`

Befehle im Hintergrund

Mit einem einzelnen `&` werden Befehle im Hintergrund ausgeführt, z. B.

```
cmd1 &
```

Das heißt, man kann direkt neue Befehle in der Shell ausführen, auch wenn `cmd1` noch nicht beendet ist.

Kommandos im Zusammenhang mit Hintergrundprozessen:

`jobs`	Befehle im Hintergrund auflisten,
`fg %nr`	Hintergrundprozess mit angegebener Job-Nr. in den Vordergrund holen,
Ctrl-Z	Aktuellen Vordergrundprozess schlafen legen (der arbeitet dann nicht mehr!),
`bg`	Schlafenden Prozess im Hintergrund weiter ausführen (so dass er wieder weiterarbeitet).

Pipes

Die Bash kennt drei Standard-Dateideskriptoren:

Standard In (stdin) Standard-Eingabe, i. A. die Tastatur,

Standard Out (stdout) Standard-Ausgabe, i. A. das aktuelle Terminal,

Standard Error (stderr) Standard-Fehler, i. A. auch das aktuelle Terminal.

Umleiten von Ein- und Ausgabe

- `wc -w < hallo.txt`
  Inhalt von „hallo.txt“ als stdin für `wc -w` (Zählen der Wörter) verwenden.
- `cat hallo.txt | wc -w`
  Inhalt von „hallo.txt“ nach stdout ausgeben, stdout aber als stdin für `wc -w` verwenden.
- `echo "Hallo Welt" > hallo.txt`
  Ausgabe in Datei „hallo.txt“ schreiben (evtl. alter Inhalt wird überschrieben).
- `echo "Hallo Welt" >> hallo.txt`
  Ausgabe an Datei „hallo.txt“ anhängen.
- `gcc -c test.c 2> error.log`
  Fehlermeldungen in Datei schreiben.
- `cmd1 | tee dateiname`
  Ausgabe parallel in Datei loggen.

Wildcards, Backticks

`*`	Beliebig viele (auch null) Zeichen,
`?`	genau ein (beliebiges) Zeichen,
`[afz1]`	eines der angegebenen Zeichen,
`[a-f]`	eines der Zeichen aus dem angegebenen Bereich,
`` `cmd` ``, `${cmd}`	wird durch Ausgabe von `cmd` ersetzt,

Variablen

Die Definition von Variablen erfolgt mit
`var=value` (ohne Leerzeichen)
und der Zugriff mit `$var`.

Zu Beachten ist die unterschiedliche Expansion von Variablen innerhalb von Anführungszeichen, z. B.

```
name=Klaus
echo "Hallo $name"
echo 'Hallo $name'
```

Im ersten Fall wird die Variable expandiert, im zweiten Fall nicht.

Globale und lokale Variablen

Die Unterscheidung zwischen lokalen und globalen Variablen in der bash ist sehr missdeutig, denn diese betrifft lediglich die Frage, ob Variablen an Subshells weitergegeben werden. Siehe dazu folgendes Beispiel:

```
varglobal="Foo"
export varglobal
varlokal="Bar"

echo $varglobal
echo $varlokal

bash
echo $varglobal
echo $varlokal
exit
```

Parametersubstitution

`${var:-default}` Der Inhalt von `$var`, falls diese nicht leer ist, sonst `default`,

`${var#pattern}` Der Inhalt von `$var`, wobei am Anfang ggfs. `pattern` entfernt wird (bei Wildcards kürzestmöglich),

`${var##pattern}` dto., aber bei Wildcards größtmögliche Entfernung von `pattern`,

`${var%pattern}` Der Inhalt von `$var`, wobei am Ende ggfs. `pattern` entfernt wird (bei Wildcards kürzestmöglich),

`${var%%pattern}` dto., aber bei Wildcards größtmögliche Entfernung von `pattern`,

Kontrollstrukturen

Die bash kennt die Kontrollstrukturen, die auch in anderen Programmiersprachen üblich sind:

- `if`,
- `while`,
- `for`,
- `case`.

Besonderheit ist, dass diese Kontrollstrukturen jeweils durch den umgedrehten Namen beendet werden, `case` z. B. durch `esac`.

Shell-Skripte in einer Datei

Ein Shell-Skript kann auch in eine Datei geschrieben werden. Ubnterscheide dabei
`./myscript` von
`. ./myscript`:

Ersteres wird in einer Subshell ausgeführt, letzteres in der aktuellen Shell. *Achtung:* Das aktuelle Verzeichnis ist meist nicht in `$PATH` enthalten, daher der Aufruf `./myscript`!

Damit das Skript direkt ausgeführt werden kann, muss es für den aktuellen User executable (Zugriffsrecht `x`) sein.

Wie wird eine Datei ausgeführt

In Linux haben Dateiendungen i. A. keine spezifische Bedeutung. Eine ausführbare Datei kann daher sowohl ein kompiliertes Programm sein, aber auch ein Skript, das mit der Bash oder aber einem Interpreter wie Perl oder Python ausgeführt werden kann.

Woher weiß Linux nun, womit ein solches Skript ausgeführt werden soll?

Hierfür existiert ein besonderer Kommentar in der ersten Zeile, durch den angegeben wird, in welchem Interpreter das Skript laufen soll, z. B.

`#! /bin/bash`

`#! /usr/bin/perl`

`#! /usr/bin/python` oder auch

`#! /usr/bin/env python` (womit `python` in `$PATH` gesucht wird).