Workstation Linuxkurs

Paul Hänsch

25. Februar 2010

1 Tag 4

1.1 Arbeit mit der Shell

Wir haben bisher einige Befehle kennen gelernt, um mit Dateien umzugehen. Von unserer Kommandozeile aus haben wir externe Programme, wie z.B. '`touch`', '`ls`' und '`chmod`' aufgerufen. Das komplexeste Programm, das wir dabei benutzt haben, war allerdings die Kommandozeilenumgebung selbst.

Die Befehlszeile wird auch als "Shell" (Schale) bezeichnet, da sie gewissermaßen das Betriebssystem umschließt, und dem Benutzer präsentiert. Um die Arbeit der Shell zu verstehen müssen wir uns unsere bisherigen Befehlsaufrufe genauer angucken:

```
ls -l -h /home/user
```

Hier wird ein Programm aufgerufen, und es werden Parameter übergeben:

Programmname	**Parameter 1**	**Parameter 2**	**Parameter 3**
`ls`	`-l`	`-h`	`/home/user`

Die Shell ist dafür verantwortlich, das eingegebene Kommando anhand der Leerzeichen zu zerlegen, die Programmdatei aufzufinden, auszuführen und dabei die Parameter einzeln zu übergeben. Wie wir sehen handelt es sich bei Parameter 3 um eine Pfadangabe. Für die Shell ist dies zunächst egal (auch wenn sie uns z.T. besondere Unterstützung bei der Eingabe von Pfaden bietet). Die Shell übergibt 3 Parameter an das Programm, wie das Programm (in diesem fall '`ls`') diese Parameter interpretiert ist nicht mehr Sache der Shell.

Bei Parameter 1 ('-l') und Parameter 2 ('-h') handelt es sich offensichtlich nicht um Pfadangaben. Dies sind **Optionen** für das Programm. Auch hier ist die Interpretation der Optionen Sache des Programms. Die Shell übergibt die Parameter nur. Nehmen wir an, wir hätten beim Aufruf von '`ls`' die Kurzform für die Optionen verwendet:

```
ls -lh /home/user
```

Die Shell hätte das Programm auf folgende Weise aufgerufen:

Programmname	**Parameter 1**	**Parameter 2**
`ls`	`-lh`	`/home/user`

Wir sehen wieder, dass nun beide Optionen als ein Parameter übergeben werden: das Programm ist dafür verantwortlich die Bedeutung dieser verkürzten Optionskette zu erkennen. Nicht jedes Programm ist dazu in der Lage, Parameter in dieser Kurzzform zu erkennen. Generell ist das Identifizieren von Optionen (z.B. anhand eines vorangehenden Bindestrichs) Sache des Programms.

Ein noch komplexeres Beispiel für die Arbeit der Shell ist das am Tag 1 ausgeführte '`echo`':

```
echo Hallo Welt > datei1
```

Programmname	**Parameter 1**	**Parameter 2**	**Shellumleitung**	**Umleitungsziel**
`echo`	`Hallo`	`Welt`	`>`	`datei1`

Zwei Dinge sind hier interssant:

1. Die Shell betrachtet die Worte '`Hallo Welt`' als zwei unterschiedliche Parameter da sie ja durch ein Leerzeichen getrennt sind.
2. Die Umleitung der Ausgabe in die Datei 'datei1' wird von der **Shell** ausgeführt, **nicht vom Programm 'echo'**. Tatsächlich erfährt das Programm '`echo`' zu keinem Zeitpunkt, dass diese Umleitung überhaupt stattfindet. Der Umleitungspfeil und das Umleitungsziel werden **nicht** als Parameter an das Programm übergeben.

Diese zweite Aussage ist besonders wichtig und wir werden gleich nochmal darauf zurückkommen. Zunächst aber nochmal zur 1. Aussage: "Hallo" und "Welt" sind zwei verschiedene Parameter. Für die Shell ist es dabei egal, wie viele Leerzeichen vor, zwischen oder nach den Parametern stehen (sie muss die Worte nur unterscheiden können). Was passiert dann aber, wenn wir z.B. "`Hallo  Welt`" (mit 10 Leerzeichen) ausgeben wollen?

```
echo Hallo          Welt
```

Programmname	Parameter 1	Parameter 2
echo	Hallo	Welt

Ausgabe ⇒

```
Hallo Welt
```

Das Programm erfährt nie, dass diese Leerzeichen existieren. Bei beiden Aufrufvarianten wurde an '`echo`' nie ein Leerzeichen übergeben (aus Sicht von '`echo`' waren die beiden letzten Aufrufe sogar identisch). Das Programm hat nur deswegen ein Leerzeichen mit ausgegeben, weil es sich "denken" konnte (bzw. weil sich sein Entwickler denken konnte), dass zwischen den beiden Parametern bei der Eingabe ein Leerzeichen stehen musste. Wie übergeben wir die Leerzeichen mit?

```
echo 'Hallo          Welt'
```

Programmname	Parameter 1
echo	Hallo Welt

Ausgabe ⇒

```
Hallo          Welt
```

- Wir haben die Zeichenkette hier bei der Befehlseingabe in einfache Anführungsstriche eingeschlossen.
- Das Programm '`echo`' erhält nur einen einzigen Parameter
- Der Parameter enthält **genau das**, was in den Anführungszeichen eingeschlossen ist, dies kann sogar Zeilenumbrüche mit einschließen (→ Ausprobieren)
- Die Anführungszeichen selbst sind **nicht** Teil des übergeben Parameters!

Aus diesem letzten Punkt entsteht übrigens wieder ein Problem: was, wenn wir die Zeichenkette "`'Hallo Welt'`" (mit Anführungszeichen) ausgeben wollen?

```
echo \'Hallo Welt\'
```

Programmname	Parameter 1	Parameter 2
echo	'Hallo	Welt'

Ausgabe ⇒

```
'Hallo Welt'
```

Wir haben jedem Anführungszeichen einfach einen Rückschrägstrich (Backslash) vorangestellt. Ein Backslash macht der Shell klar, dass sie das jeweils nachfolgende Zeichen nicht selbst interpretieren, sondern einfach ans Programm "durchreichen" soll. Man bezeichnet den Backslash in dieser Funktion als "Escapecharacter" oder deutsch "Fluchtzeichen" (umgangssprachlich oft "Escapezeichen") da ein Zeichen damit der Interpretation durch die Shell "entflieht". das Fluchtzeichen selbst wirrd nicht ans Programm ubergeben. Wenn wir einen Backslash selbst übergeben wollen, "escapen" wir diesen übrigens einfach auch mit einem Backslash:

```
echo \\
⇒
\
```

1.2 Shellumleitungen

Wir erwähnten vorher die Shellumleitung in Form eines Pfeils. Für uns sind zunächst zwei Arten von Shellumleitungen interessant:

- die Umleitung von Programmausgaben in eine Datei (`>` und `>>`)
- die Umleitung von Programmausgaben an ein weiteres Programm (`|`)

1.2.1 Ausgabeumleitung in Dateien

Die Umleitung in eine Datei ist uns bereits bekannt. Wir haben sie wiederholt beim Aufruf von 'echo' durchgeführt.

```
echo Hallo Welt > datei1
```

Da die Umleitung von der Shell ausgeführt wird, ist sie vollkommen Unabhängig vom Programm. Wir können damit also die Ausgabe von jedem beliebigen Programm in eine Datei leiten:

```
ls > datei1
cat datei1
⇒
... (Verzeichnisauflistung)
```

Angenommen, wir tun folgendes:

```
ls > datei1
echo Hallo Welt > datei1
cat datei1
⇒
Hallo Welt
```

Die Ausgabe von 'ls' ist nicht mehr in "datei1" enthalten, denn die Datei wurde bei der zweiten Ausgabenumleitung (hinter 'echo') überschrieben. Mit dem Doppelpfeil können wir eine umgeleitete Ausgabe an die Datei anhängen:

```
ls > datei1
echo Hallo Welt >> datei1
cat datei1
⇒
... (Verzeichnisauflistung)
Hallo Welt
```

1.2.2 Ausgabeumleitung an Programme

Sehen wir uns zunächst zwei weitere Unix-Befehle an, mit denen diese Art der Umleitung sinnvoll ist:

- **wc** - "word count" ("Wörterzahl") gibt die Anzahl von Zeilen, Wörtern und Zeichen in einer angegebenen Datei aus
- **tr** - "translate" (übersetze) ersetzt Zeichen in einem Datenstrom durch andere.

Das Kommando 'wc' kann auf Dateien angesetzt werden, um darin enthaltene Zeilen, Wörter und Zeichen zu zählen:

```
echo Hallo Welt > datei1
echo Was geht ab? >> datei1
wc datei1
⇒
 2  5 24 datei1
```

Die Datei hat demnach 2 Zeilen, 5 Wörter und insgesamt 24 Zeichen. Was ist nun, wenn wir eine solche Statistik z.B. für die Ausgabe von 'ls' haben wollen? Das Programm 'wc' ist in der Lage, seinen Text nicht nur aus einer Datei, sondern auch aus einer Umleitung zu lesen:

```
ls |wc
⇒
     72      89     992
```

Die angezeigten Zahlenwerte hängen natürlich davon ab, welches Verzeichnis wir uns von 'ls' ausgeben lassen. In diesem Fall haben wir 72 Zeilen ('ls' listet genau 1 Objekt pro Zeile), 89 Wörter (einige Dateinamen hier enthalten also offenbar Leerzeichen) und insgesamt 992 Zeichen. Gucken wir uns die Zeile nochmal genauer an:

Programmname	**Shellumleitung**	**Programmname**
`ls`	`|`	`wc`

Wir haben also zwei Programme gleichzeitig ausgeführt und dabei die Programmausgabe des ersten Programms ('ls') zur Eingabe des zweiten Programms ('wc') gemacht.

Einige Programme sind sogar darauf ausgelegt, nur auf diese Weise zu arbeiten, und gar nicht mit Dateien umzugehen. ein solches Programm ist '`tr`':

```
echo Der Esel lauert im Teich. |tr s g
⇒
Der Egel lauert im Teich.
```

Das Programm '`tr`' hat aus unserem Esel einen Egel gemacht. Betrachten wir kurz den Informationsfluss, der dabei Stattgefunden hat. Jedes Programm hat beim Start mindestens 3 Kanäle zum Austausch von Informationen:

- **stdin** "Standard Input" ist der Kanal, auf dem Ein Programm Informationen liest.
- **stdout** "Standard Output" ist der Kanal, auf dem Ein Programm Informationen ausgibt.
- **stderr** "Standard Error" ist der Kanal, auf dem Ein Programm Fehlerausgaben schreibt.

Normalerweise ist 'stdin' eine Eingabe, die von der Tastatur gelesen wird, während 'stdout' und 'stderr' auf dem Terminal angezeigt werden. Die Shell ist in der Lage, diese Kanäle "umzubiegen". Betrachten wir den Befehl von eben noch einmal genau:

stdin von echo	**Programm**	**Param1**	**Param2**	**Param3**	**Param4**	**Param5**	**stdout von echo**
(nichts)	`echo`	Der	Esel	lauert	im	Teich.	Der Esel lauert im Teich.

Umleitung
|

stdin von tr	**Programm**	**Param1**	**Param2**	**stdout von tr**
Der Esel lauert im Teich.	`tr`	s	g	Der Egel lauert im Teich.