1.2.F Anwendung/Befehle – Ergänzungen und Bilder

1.2.F.1 echo-Befehl

Mit dem Befehl `echo` kann man Zeichenketten ausgeben. Jeder Parameter wird als eine Zeichenkette angesehen; die Zeichenketten sind in der Ausgabe durch jeweils ein Leerzeichen voneinander getrennt:

Terminal
```
schueler@debian964:~$ echo Das ist das Haus vom           Nikolaus.
Das ist das Haus vom Nikolaus.
```

Will man den obigen Satz genau wie eingegeben ausgeben, dann muss man ihn davor schützen, dass er durch die Shell in mehrere Parameter getrennt wird (*word splitting*). Man nennt diesen Schutz *Maskierung*. Man kann die Zeichenkette maskieren, indem man sie in Anführungszeichen setzt. Sie wird dann als ein einziger Parameter an den Befehl `echo` übergeben:

Terminal
```
schueler@debian964:~$ echo "Das ist das Haus vom           Nikolaus."
Das ist das Haus vom           Nikolaus.
```

Mit der Option `-n` (*no linefeed*) kann man den abschließenden Zeilenumbruch verhindern:

Terminal
```
schueler@debian964:~$ echo -n Quak
Quakschueler@debian964:~$
```

Mit der Option `-e` kann man Escape-Sequenzen interpretieren lassen. Da sind z. B. die von Java und C bekannten Ersatzdarstellungen wie `\n` und `\r`:

Terminal
```
schueler@debian964:~$ echo -e "1 \0342\0202\0254\n2 \0342\0202\0254"
1 €
2 €
```

Darüberhinaus gibt es Escape-Sequenzen für die Terminal-Steuerung. Viele von ihnen beginnen mit dem ASCII-Zeichen Nr. 27 (Escape). Man kann es oktal als `\033` eingeben oder einfach als `\e`:

Terminal
```
schueler@debian964:~$ echo -e "Text in \e[31mrot."
Text in rot.
```

Nicht nur der Befehl `echo` kann Escape-Sequenzen ersetzen, sondern auch die Bash selbst (logisch, weil `echo` ein eingebauter Befehl ist). Dazu gibt es eine eigene Schreibweise:

Terminal
```
schueler@debian964:~$ echo Eins$'\n'Zwei
Eins
Zwei
```

Hier kommt `echo` ohne die Option `-e` aus – die Ersetzung wird bereits vor dem Aufruf von `echo` durch die Shell vorgenommen. Auf diese Art kann man beliebige Zeichenketten mit Sonderzeichen zusammenbasteln, ohne an den `echo`-Befehl gebunden zu sein.

Auf alten Systemen kann es manchmal vorkommen, dass man sich durch die Ausgabe einer Binärdatei, die zufällig Terminalsequenzen enthält, seinen Bildschirmfont verstellt. Dann hilft das Programm `reset`. Hat man das nicht, gibt man das ASCII-Zeichen Nr. 15 aus (eventuell blind), und der alte Font ist wieder da:

Terminal
```
schueler@debian964:~$ echo -e "\017"
```

1.2.F.2 Befehle für Dateien und Verzeichnisse

Tabelle 1 zeigt eine Übersicht, welche Befehle für übliche Aktionen an Dateien, leeren Verzeichnissen und an ganzen Teilbäumen (=Verzeichnissen mit Inhalten) geeignet sind.

Aktion	Datei	Leeres Verzeichnis	Verzeichnis mit Inhalt
Anlegen (leer)	`touch x`	`mkdir x`	`mkdir x x/y x/z`
Löschen	`rm x`	`rmdir x`	`rm -r x`
Verschieben	`mv x y`	`mv x y`	`mv x y`
Kopieren	`cp x y`	`cp -r x y`	`cp -r x y`
Eigenschaften	`stat x`	`stat x`	`du x`

Tabelle 1: Übersicht über Befehle für Dateien und Verzeichnisse

1.2.F.3 `cp`, `mv` und das Zielobjekt

Befehlszeile	z existiert nicht	z ist Datei	z ist Verzeichnis
cp d1 z	anlegen z	überschreiben z	kopieren nach z/d1
cp d1 z/	Fehlermeldung	Fehlermeldung	"
cp d1 d2 z	"	"	kopieren nach z/d1 z/d2
cp d1 d2 z/	"	"	"
cp -r v1 z	anlegen z/	"	kopieren nach z/v1
cp -r v1 z/	anlegen z/	"	"
cp -r v1 v2 z	Fehlermeldung	"	kopieren nach z/v1 z/v2
cp -r v1 v2 z/	"	"	"
mv d1 z	umbenennen z	überschreiben z	verschieben nach z/d1
mv d1 z/	Fehlermeldung	Fehlermeldung	"
mv d1 d2 z	"	"	verschieben nach z/d1 z/d2
mv d1 d2 z/	"	"	"
mv v1 z	umbenennen z	"	verschieben nach z/v1
mv v1 z/	umbenennen z	"	"
mv v1 v2 z	Fehlermeldung	"	verschieben nach z/v1 z/v2
mv v1 v2 z/	"	"	"

Tabelle 2: Was passiert bei `cp` und `mv`?

Die Befehle zum Kopieren und Verschieben (`cp` und `mv`) versuchen herauszufinden, was der Benutzer möchte. Das kann schon bei einem einfachen Befehl wie `cp x y` mehrdeutig sein:

- Soll die Datei x in die Datei y kopiert werden? Das Ergebnis wäre die Datei y im aktuellen Verzeichnis.
- Soll die Datei x in das Verzeichnis y kopiert werden? Das Ergebnis wäre die Datei y/x im Verzeichnis y.

Um das herauszufinden, werten die beiden Programme folgende Fragen aus:

a) Bei mehr als zwei Parametern muss das Ziel ein Verzeichnis sein, denn man kann nicht zwei oder mehr Dateien in eine verschieben

b) Falls der Name des Ziels mit einem Schrägstrich `"/"` endet, muss es ein Verzeichnis sein

c) Falls der Name des Ziels ein bereits existierendes Verzeichnis ist, wird dieses als Ziel angenommen

Tabelle 2 zeigt die Ergebnisse. d1 und d2 sind Dateien, v1 und v2 Verzeichnisse.

1.2.F.4 Tastenkombinationen zur Eingabe von Sonderzeichen:

- Linux-Konsole (Alt-F1 bis Alt-F4): Alt -Num-Tasten-Dezimalzahl ergibt Zeichen gemäß Tabelle nach Unicode: Alt -321 gibt Ł. Die Ausgabe erfolgt aber ganz normal im Konsolen-Zeichensatz, und das ist UTF-8.
- GTK-Anwendungen: Shift ⇑ - Strg + u -Hexadezimalzahl ergibt Zeichen gemäß Tabelle nach Unicode: Shift ⇑ - Strg + u -141 gibt Ł.
- Zu Unicode: `http://www.unicode.org/charts/`.
- ANSI-C-Quoting: `$'\101'` expandiert überall auf der Befehlszeile zu `A` (ASCII-Zeichen Nr. 101 oktal=Nr. 65 dezimal).
  - `\nnn`: 1–3 Oktalziffern für ein Byte (siehe obiges Beispiel `$'\101'`)
  - `\xHH`: 1–2 Hexadezimalziffern für ein Byte (anders als in C/C++)
  - `\uHHHH`: 1–4 Hexadezimalziffern für einen Unicode-Punkt (anders als in C/C++)
  - `\UHHHHHHHH`: 1–8 Hexadezimalziffern für einen Unicode-Punkt (anders als in C/C++)
  - Ersatzdarstellungen wie in C/C++: `\a` (Alarm), `\b` (Backspace), `\e` und `\E` (Escape, nicht von C/C++), `\f` (Seitenvorschub), `\n` (Zeilenvorschub), `\r` (Wagenrücklauf), `\t` (Tabulator), `\v` (Vertikaltabulator), `\\` (Backslash), `\'` (Hochkomma), `\"` (Gänsefüßchen), `\?` (Fragezeichen)
  - Steuerzeichen 0 bis 31:
    `\cx`: Strg-x mit x von A=1 bis Z=26, außerdem `@`=0 `[`=27, `\`=28, `]`=29, `^`=30, `_`=31