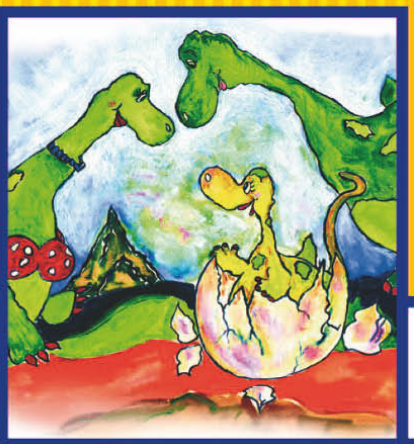
COMPANION WEBSITE
CWS
www.pearson-studium.de



st
scientific tools

Michael Kofler
Jürgen Plate

Linux für Studenten

PEARSON
Studium

Michael Kofler
Jürgen Plate

Linux für Studenten

ein Imprint von Pearson Education
München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

export [optionen] variable [=wert]

export deklariert die angegebene Shell-Variable als Umgebungs-Variable. Damit ist die Variable auch in allen aufgerufenen Kommandos und Subshells verfügbar. Optional kann dabei auch eine Variablenzuweisung erfolgen. Wenn das Kommando ohne Parameter aufgerufen wird, werden alle Umgebungs-Variablen angezeigt.

-n macht eine Umgebungs-Variable wieder zu einer normalen Shell-Variablen. Das Kommando hat damit genau die umgekehrte Wirkung wie bei der Verwendung ohne Optionen.

expr zeichenkette : muster

expr kann zur Auswertung arithmetischer Ausdrücke, zum Vergleich zweier Zeichenketten etc. eingesetzt werden. Von test und dem Substitutionsmechanismus $[...] unterscheidet sich expr insofern, als es in der oben angeführten Syntaxvariante Mustervergleiche für Zeichenketten durchführen kann (reguläre Ausdrücke).

file [optionen] datei

file versucht festzustellen, welchen Datentyp die als Parameter angegebene Datei hat. Als Ergebnis liefert file eine Zeichenkette mit dem Dateinamen und dem Typ der Datei. *Achtung*: Textdateien mit deutschen Sonderzeichen werden nicht als Textdateien klassifiziert, sondern als data. Die Klassifizierung basiert auf der Datei /etc/magic. file -z datei versucht den Datentyp einer komprimierten Datei zu erkennen.

```
for var [in liste;] do
    kommandos
done
```

for bildet eine Schleife. In die angegebene Variable werden der Reihe nach alle Listenelemente eingesetzt. Die Liste kann auch mit Jokerzeichen für Dateinamen oder mit {...}-Elementen zur Zusammensetzung von Dateinamen gebildet werden. Wenn auf die Angabe der Liste verzichtet wird, durchläuft die Variable alle der Shell-Datei übergebenen Parameter (also in $*).

```
[ function ] name()
    { kommandos }
```

function definiert eine Subfunktion, die innerhalb der Shell-Datei wie ein neues Kommando aufgerufen werden kann. Innerhalb der Funktion können mit local lokale Variablen definiert werden. Funktionen können rekursiv aufgerufen werden. Den einzelnen Funktionen können Parameter wie Kommandos übergeben werden. Innerhalb der Funktion können diese Parameter den Variablen $1, $2 etc. entnommen werden.

```
if bedingung; then
    kommandos
[elif bedingung; then
    kommandos]
```

```
[else
     kommandos]
fi
```

Das `if`-Kommando leitet eine Verzweigung ein. Der Block nach `then` wird nur ausgeführt, wenn die Bedingung erfüllt ist. Andernfalls werden (beliebig viele, optionale) `elif`-Bedingungen ausgewertet. Gegebenenfalls wird der ebenfalls optionale `else`-Block ausgeführt.

Als Bedingung können mehrere Kommandos angegeben werden. Nach dem letzten Kommando muss ein Semikolon folgen. Als Kriterium gilt der Rückgabewert des letzten Kommandos. Vergleiche und andere Tests können mit dem Kommando `test` durchgeführt werden. Statt `test` ist auch eine Kurzschreibweise in eckigen Klammern zulässig, dabei muss aber nach `[` und vor `]` jeweils ein Leerzeichen stehen.

`local var[=wert]`

`local` definiert eine lokale Variable. Das Kommando kann nur innerhalb einer selbst definierten Funktion verwendet werden (siehe `function`). Vor und nach dem Gleichheitszeichen dürfen keine Leerzeichen angegeben werden.

`popd`

`popd` wechselt in ein zuvor mit `pushd` gespeichertes Verzeichnis zurück. Das Verzeichnis wird aus der Verzeichnisliste entfernt.

`printf format para1 para2 para3 ...`

`printf` erlaubt es, Ausgaben in der Syntax des C-Kommandos *printf* zu formatieren. Detaillierte Informationen zu den Formatierungsmöglichkeiten erhalten Sie mit `man 3 printf`.

`pushd verzeichnis`

`pushd` speichert das aktuelle Verzeichnis und wechselt anschließend in das angegebene Verzeichnis. Mit `popd` kann in das ursprüngliche Verzeichnis zurückgewechselt werden. `dirs` zeigt die Liste der gespeicherten Verzeichnisse an.

`read [var1 var2 var3 ...]`

`read` liest eine Zeile Text in die angegebenen Variablen. `read` erwartet die Daten aus der Standardeingabe. Wenn keine Variable angegeben wird, schreibt `read` die Eingabe in die Variable `REPLY`. Wenn genau eine Variable angegeben wird, schreibt `read` die gesamte Eingabe in diese eine Variable. Wenn mehrere Variablen angegeben werden, schreibt `read` das erste Wort in die erste Variable, das zweite Wort in die zweite Variable ... und den verbleibenden Rest der Eingabe in die letzte Variable. Wörter werden dabei durch Leer- oder Tabulatorzeichen getrennt.

Das `read`-Kommando sieht keine Möglichkeit vor, einen Infotext als Eingabeaufforderung auszugeben. Deswegen ist es zweckmäßig, den Anwender vor der Ausführung von `read`-Kommandos mit `echo -n` über den Zweck der Eingabe zu informieren.

setterm [option]

setterm verändert diverse Einstellungen des Terminals. Wenn das Kommando ohne die Angabe einer Option ausgeführt wird, zeigt es eine Liste aller möglichen Optionen an. Nützliche Optionen zur Shell-Programmierung sind:

- **-bold on** | **off** aktiviert bzw. deaktiviert die fette Schrift. In Textkonsolen erscheint der Text zwar nicht fett, aber immerhin in einer anderen Farbe als der sonstige Text.
- **-clear** löscht den Inhalt des Terminals.
- **-default** stellt Farben und Textattribute auf die Default-Einstellung zurück.
- **-half-bright on** | **off** stellt hervorgehobene Schrift ein/aus.
- **-reverse on** | **off** stellt inverse Schrift ein/aus.
- **-underline on** | **off** stellt unterstrichene Schrift ein/aus.

shift [n]

shift schiebt die dem Shell-Programm übergebene Parameterliste durch die vordefinierten Variablen $1 bis $9. Wenn shift ohne Parameter verwendet wird, werden die Parameter um eine Position verschoben, andernfalls um *n* Positionen. shift ist besonders dann eine wertvolle Hilfe, wenn mehr als neun Parameter angesprochen werden sollen. *Achtung*: Einmal mit shift aus den Variablen geschobene Parameter können nicht mehr angesprochen werden. Sie werden auch aus der Variablen $* entfernt.

sleep zeit

sleep versetzt das laufende Programm für die angegebene Zeit in den Ruhezustand. Das Programm konsumiert in dieser Zeit praktisch keine Rechenzeit. Die Zeitangabe erfolgt normalerweise in Sekunden. Optional können die Buchstaben m, h oder d angehängt werden, um die Zeit in Minuten, Stunden oder Tagen anzugeben.

source datei

source führt die angegebene Shell-Datei so aus, als befänden sich die darin enthaltenen Kommandos an der Stelle des source-Kommandos. Nach der Ausführung der Datei wird das laufende Shell-Programm in der nächsten Zeile fortgesetzt. Zur Ausführung der angegebenen Datei wird keine neue Shell gestartet. Alle Variablen (inklusive der Parameterliste) gelten daher auch für die angegebene Datei. Wenn in dieser Datei exit ausgeführt wird, kommt es *nicht* zu einem Rücksprung in das Programm mit dem source-Kommando, sondern zu einem sofortigen Ende der Programmausführung. Zu source existiert die Kurzform . datei.

test ausdruck

test wird zur Formulierung von Bedingungen verwendet und zumeist in if-Abfragen und Schleifen eingesetzt. Je nachdem, ob die Bedingung erfüllt ist, liefert es den Wahrheitswert 0 (wahr) oder 1 (falsch). Statt test kann auch die Kurzschreibweise

`[ ausdruck ]` verwendet werden, wobei Leerzeichen vor und nach dem Ausdruck angegeben werden müssen.

Wenn *test* oder die Kurzschreibweise *[ausdruck]* als Bedingung in einer Verzweigung oder Schleife verwendet wird, muss die Bedingung mit einem Semikolon abgeschlossen werden, also z. B. `if [ "$1" = "abc" ]; then ...`

if-Abfragen können manchmal durch die Formulierung `test "$1" = "abc" && kommando` ersetzt werden. In diesem Fall ist kein Semikolon erforderlich. Das Kommando wird nur ausgeführt, wenn die vorherige Bedingung erfüllt war.

Zeichenketten

`[ zk ]`	wahr, wenn die Zeichenkette nicht leer ist
`[ -z zk ]`	wahr, wenn die Zeichenkette leer ist (0 Zeichen)
`[ zk1 = zk2 ]`	wahr, wenn die Zeichenketten übereinstimmen
`[ zk1 != zk2 ]`	wahr, wenn die Zeichenketten voneinander abweichen

Die Zeichenketten bzw. Variablen sollten in Hochkommata gestellt werden (z. B. `[ "$1" = "abc" ]` oder `[ "$a" = "$b" ]`). Andernfalls kann es bei Zeichenketten mit mehreren Wörtern zu Fehlern kommen.

Zahlen

`[ z1 -eq z2 ]`	wahr, wenn die Zahlen gleich sind (equal)
`[ z1 -ne z2 ]`	wahr, wenn die Zahlen ungleich sind (not equal)
`[ z1 -gt z2 ]`	wahr, wenn *z1* größer *z2* ist (greater than)
`[ z1 -ge z2 ]`	wahr, wenn *z1* größer gleich *z2* ist (greater equal)
`[ z1 -lt z2 ]`	wahr, wenn *z1* kleiner *z2* ist (less than)
`[ z1 -le z2 ]`	wahr, wenn *z1* kleiner gleich *z2* ist (less equal)

Dateien (auszugsweise)

`[ -d dat ]`	wahr, wenn es sich um ein Verzeichnis handelt (directory)
`[ -e dat ]`	wahr, wenn die Datei existiert (exist)
`[ -f dat ]`	wahr, wenn es sich um eine einfache Datei (und nicht um ein Device, ein Verzeichnis ...) handelt (file)
`[ -L dat ]`	wahr, wenn es sich um einen symbolischen Link handelt
`[ -r dat ]`	wahr, wenn die Datei gelesen werden darf (read)
`[ -s dat ]`	wahr, wenn die Datei mindestens 1 Byte lang ist (size)
`[ -w dat ]`	wahr, wenn die Datei verändert werden darf (write)
`[ -x dat ]`	wahr, wenn die Datei ausgeführt werden darf (execute)
`[ dat1 -ef dat2 ]`	wahr, wenn beide Dateien denselben I-Node haben (equal file)
`[ dat1 -nt dat2 ]`	wahr, wenn Datei 1 neuer als Datei 2 ist (newer than)

Verknüpfte Bedingungen

`[ ! bed ]`	wahr, wenn die Bedingung nicht erfüllt ist
`[ bed1 -a bed2 ]`	wahr, wenn beide Bedingungen erfüllt sind (and)
`[ bed1 -o bed2 ]`	wahr, wenn mindestens eine der Bedingungen erfüllt ist (or)

`trap [kommando] n`

`trap` führt das angegebene Kommando aus, wenn in der `bash` das angegebene Signal auftritt. Wenn kein Kommando angegeben wird, ignoriert das Programm bzw. die `bash` das betreffende Signal. `trap -l` liefert eine Liste aller möglichen Signale und der ihnen zugeordneten Kenn-Nummern.

`unalias abkürzung`

`unalias` löscht eine vorhandene Abkürzung. Wenn das Kommando mit der Option `-a` aufgerufen wird, löscht es alle bekannten Abkürzungen.

`unset variable`

`unset` löscht die angegebene Variable.

`until bedingung; do`

```
    kommandos
done
```

`until` dient zur Bildung von Schleifen. Die Schleife wird so lange ausgeführt, wie die angegebene Bedingung erfüllt ist. Das Schleifenkriterium ist der Rückgabewert des Kommandos, das als Bedingung angegeben wird. Vergleiche und Tests werden mit dem Kommando `test` oder dessen Kurzform in eckigen Klammern durchgeführt.

`wait [prozessnummer]`

`wait` wartet auf das Ende des angegebenen Hintergrundprozesses. Wenn keine Prozessnummer angegeben wird, wartet das Kommando auf das Ende aller laufenden, von der Shell gestarteten Hintergrundprozesse.

`while bedingung; do`

```
    kommandos
done
```

`while` dient zur Bildung von Schleifen. Die Schleife wird so lange ausgeführt, bis die angegebene Bedingung zum ersten Mal nicht mehr erfüllt ist. Das Schleifenkriterium ist der Rückgabewert des Kommandos, das als Bedingung angegeben wird. Vergleiche und Tests werden mit dem Kommando `test` oder dessen Kurzform in eckigen Klammern durchgeführt.

4.11 Referenz aller Sonderzeichen

`;`	trennt mehrere Kommandos
`:`	Shell-Kommando, das nichts tut
`.`	Shell-Programm ohne eigene Subshell starten (`. datei`) (entspricht `source datei`)
`#`	leitet einen Kommentar ein
`#!/bin/sh`	identifiziert die gewünschte Shell für das Shell-Programm
`&`	führt das Kommando im Hintergrund aus (`kom &`)
`&&`	bedingte Kommando-Ausführung (`kom1 && kom2`)

`&>`	Umleitung von Standardausgabe und -fehler (entspricht `>&`)
`|`	bildet Pipes (`kom1 | kom2`)
`||`	bedingte Kommando-Ausführung (`kom1 || kom2`)
`*`	Jokerzeichen für Dateinamen (beliebig viele Zeichen)
`?`	Jokerzeichen für Dateinamen (ein beliebiges Zeichen)
`[abc]`	Jokerzeichen für Dateinamen (ein Zeichen aus `abc`)
`[ ausdruck ]`	Kurzschreibweise für `test ausdruck`
`(...)`	Kommandos in derselben Shell ausführen (`(kom1; kom2)`)
`{...}`	Kommandos gruppieren
`~`	Abkürzung für das Heimatverzeichnis
`>`	Ausgabeumleitung in eine Datei (`kom > dat`)
`>>`	Ausgabeumleitung; an vorhandene Datei anhängen
`>&`	Umleitung von Standardausgabe und -fehler (entspricht `&>`)
`2>`	Umleitung der Standardfehlerausgabe
`<`	Eingabeumleitung aus einer Datei (`kom < dat`)
`<< ende`	Eingabeumleitung aus der aktiven Datei bis zu `ende`
`$`	Kennzeichnung von Variablen (`echo $var`)
`$!`	PID des zuletzt gestarteten Hintergrundprozesses
`$$`	PID der aktuellen Shell
`$0`	Dateiname des gerade ausgeführten Shell-Scripts
`$1` bis `$9`	die ersten neun dem Kommando übergebenen Parameter
`$#`	Anzahl der dem Shell-Programm übergebenen Parameter
`$*` oder `$@`	Gesamtheit aller übergebenen Parameter
`$?`	Rückgabewert des letzten Kommandos (0 = OK oder Fehlernummer)
`$(...)`	Kommandosubstitution (`echo $(ls)`)
`${...}`	diverse Spezialfunktionen zur Bearbeitung von Zeichenketten
`$[...]`	arithmetische Auswertung (`echo $[2+3]`)
`"..."`	Auswertung der meisten Sonderzeichen verhindern
`'...'`	Auswertung aller Sonderzeichen verhindern
blabla	nur als Trenner
`‘...‘`	Kommandosubstitution (`echo ‘ls‘`)
`\zeichen`	hebt die Wirkung des Sonderzeichens auf

4.12 Aufgaben

Die Lösungen zu diesen Aufgaben finden Sie im Anhang C.1.

1. Es geht um Kommandosubstitution. Was beinhaltet nach Abarbeitung der Kommandofolge die Datei `datei3`?

```
echo "blau" > kunde
echo "gruen" > kun.de
echo "gelb" >> kunde
echo "rot" > kunde.r
echo kund* >datei2
cat ‘cat datei2‘ > datei3
```

2. Mit Pipes kann man recht komfortable Kommandos bilden. Verwenden Sie die Hintereinanderschaltung von `ps -ef` und `grep`, um die Prozesse eine bestimmten Benutzers (z. B. root) oder eines bestimmten Terminals anzuzeigen.
3. Geben Sie einen Aufruf von `find` an, der im Verzeichnis `/home` alle Dateien sucht, auf die innerhalb der letzten 10 Tage zugegriffen wurde.
4. Welches `find`-Kommando müsste man verwenden, um in allen Dateien auf der Platte mit der Endung „.txt“ nach der Zeichenkette „UNIX“ zu suchen.
5. Schreiben Sie ein Shell-Skript, das in einer Endlosschleife alle 10 Sekunden die Uhrzeit (Stunde und Minute) per `banner`-Kommando auf den Bildschirm ausgibt und vorher den Bildschirm löscht.
6. Schreiben Sie ein Shell-Skript „ggt“, das den größten gemeinsamen Teiler der als Parameter übergebenen beiden Zahlen berechnet. Formulieren Sie die Berechnung des GGT als Shell-Funktion. Der Algorithmus lautet folgendermaßen:

```
ggt(x,y):
solange x ungleich y ist, wiederhole
  falls x > y  dann  x = x - y
  sonst              y = y - x;
```

   Denken Sie daran, dass man zum Rechnen das Kommando `expr` braucht.
7. Eine Datei namens `personal` habe folgendes Aussehen:

```
[Name]        [Vorname]     [Wohnort]     [Geb.-Datum]
=========================================================
Meyer         Peter         Berlin         10.10.1970
Schulze       Axel          Hamburg        12.12.1980
Lehmann       Rita          München        17.04.1971
```

   Sortieren Sie die Daten der Datei (ohne Zeile 1 und 2) nach dem Namen und geben Sie das Ergebnis in eine Datei `personal.sort` aus. Die beiden Überschriftszeilen sollen natürlich wieder am Dateianfang stehen.
8. Schreiben Sie ein Shellscript, das an alle Benutzer mit der Gruppennummer 100 eine E-Mail verschickt. Betreff und die Datei, welche den E-Mailtext enthält, werden als Parameter an das Script übergeben.

WWW, E-Mail, NFS, SSH und SCP

5

ÜBERBLICK

Dieses Kapitel stellt die wichtigsten Programme zum Surfen im Web, zum Lesen und Verfassen von E-Mails sowie zum Dateizugriff und -transfer vor. Unter Linux steht eine breite Palette derartiger Programme zur Verfügung. Die folgende Liste gibt Ihnen einen ersten Überblick über die wichtigsten Programme:

- Mozilla: Universalprogramm für Web, E-Mail und News
- Firefox: Webbrowser (hervorgegangen aus Mozilla)
- Thunderbird: Mail-Client (hervorgegangen aus Mozilla)
- Konqueror: KDE-Standardbrowser
- Epiphany: Gnome-Standardbrowser
- KMail: KDE-Mail-Client
- Lynx: textbasierter Browser
- Mutt: textbasierter Mail-Client

5.1 Webbrowser

Dieser Abschnitt stellt die wichtigsten unter Linux verfügbaren Webbrowser vor. Da wir davon ausgehen, dass Sie die Grundfunktionen dieser Programme auch ohne dieses Buch nutzen können, beschränken wir uns auf die Beschreibung von Besonderheiten bzw. Konfigurationsdetails.

5.1.1 Die Mozilla-Familie

Mozilla hat sich in den vergangenen Jahren zu dem Standardbrowser für Linux entwickelt. Mozilla setzt die verschiedenen Web- und HTML-Standards sehr genau um und verfügt über einen ausgereiften JavaScript-Interpreter. Die Anzahl der Websites, die mit Mozilla nicht genutzt werden können, ist daher ziemlich gering. Meist handelt es sich dabei um Sites, die nicht standardisierte, Internet-Explorer-spezifische Erweiterungen verwenden oder die Plugins bzw. ActiveX-Komponenten nutzen, die nur unter Microsoft Windows zur Verfügung stehen.

Mozilla ist ein Universalprogramm, das gleichermaßen zum Surfen im Web, zur Verwaltung von E-Mails und zum Lesen von News eingesetzt werden kann. Je nach Installation kann das Programm auch als IRC-Client, als HTML-Editor und zur Terminverwaltung verwendet werden (siehe *Tools*- bzw. *Fenster*-Menü).

Firefox und Thunderbird: Anfang 2003 beschlossen die Mozilla-Entwickler, das immer größer werdende Programm in zwei wesentliche Komponenten zu zerlegen: den neuen Webbrowser „Mozilla Firebird“ und den neuen E-Mail-Client „Mozilla Thunderbird“. Anfang 2004 wurde „Firebird“ in „Firefox“ umbenannt. Die beiden Programme haben zwar noch einige Ecken und Kanten (insbesondere Firefox), sind aber schon gut verwendbar. Es wird aber wohl noch etwas dauern, bis beide Programme wirklich ausgereift sind. Aus diesem Grund wird auch Mozilla selbst weiterentwickelt. In den kommenden Mozilla-Versionen sind allerdings kaum neue Funktionen zu erwarten. Ausführliche Informationen über alle drei Projekte finden Sie ausgehend von der Mozilla-Website *http://www.mozilla.org/*.