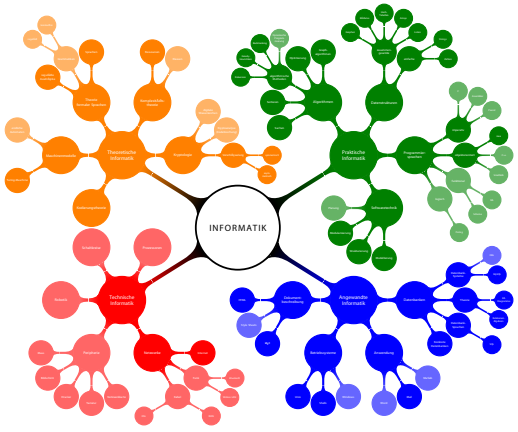
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 4

Shells

Dialog mit der Waschmaschine

Vorlesung Einführung in die Informatik 1 vom 5. November 2013 von Till Tantau



Lernziele von Kapitel 4

1. Unix- und Programm-Shells benutzen können
2. Wichtige Unix-Kommandos kennen
3. Einfache Shell-Skripte erstellen können

Gliederung von Kapitel 4

Wozu dienen Shells?

Shells stellen eine Dialog zwischen einem *Benutzer oder einem Programm* und einem *Programm* her. Der Ablauf:

1. Die Shell zeigt einen *Prompt* an.
   Der Prompt ist eine Eingabeaufforderung, eine Frage der Form »Meister, wie kann ich Euch dienen?«.
2. Der Benutzer gibt einen *Befehl* ein.
3. Die Shell führt diesen aus.
4. Danach zeigt sie wieder einen Prompt und die Sache beginnt von neuem.

Die Befehle, die *eine Shell versteht,* hängen von der Shell ab.

Viele Programme haben Shells

- Shells zum Unix-Betriebssystem-Kern:
  - `sh`,
  - `bash`,
  - `csh`,
  - `ksh`.
- Shells zum Windows-Betriebssystem-Kern:
  - `command.com`.
- Shells zu anderen Rechnern:
  - `ftp` (file transport protocol),
  - `ssh` (secure shell).
- Shells zu Datenbanken:
  - `mysql`.
- Shells zu Quake, einem Computerspiel.

Der Prompt einer Shell

Bash-Prompt

Shells melden sich mit einem *Prompt*, auch *Eingabeaufforderung* genannt.

Beispiel

Die Unix-Shell Bash meldet sich mit

```
tantau@pcclt02:~/uebeung06>
```

Dies heißt so viel wie »Hallo Benutzer `tantau`! Hier ist die Bash-Shell auf dem Rechner `pcclt02`. Du bist gerade im Verzeichnis `~/uebungs06` und ich wüsste jetzt gerne, wie es weitergehen soll.«

Der Prompt einer Shell

FTP- und MySQL-Prompts

Beispiel

Das Programm `ftp` meldet sich mit

```
ftp>
```

Dies heißt so viel wie »Hallo Benutzer, hier ist das `ftp` Programm. Wo wir gerade sind und was jetzt passieren soll musst du schon selbst wissen. Wenn du es weißt, dann sag mal Bescheid.«

Beispiel

Das Datenbank `mysql` meldet sich mit

```
mysql>
```

Dies heißt so viel wie »Hallo Benutzer, hier ist das `mysql` Programm. Ich sage auch nicht, was gerade los ist.«

Start und Benutzung einer Unix-Shell

- Zum Starten einer Unix-Shell ruft man ein *Terminal-* oder *Konsolenprogramm* auf. Dieses stellt ein Fenster dar, in dem dann eine Standard-Shell startet.
- Als Antwort auf den Prompt gibt man den Namen eines Befehls ein. (Ein Befehl ist ein kleines Programm.)
- Dieser wird dann ausgeführt und seine Ausgabe angezeigt.
- Ist der Befehl (das Programm) fertig, so erscheint wieder ein Prompt.

Beispiel

```
tantau@pcclt02:~> ls
public_html  tmp  uebeung06
tantau@pcclt02:~> pwd
/home/tantau
tantau@pcclt02:~> exit
```

Optionen bei Befehlen

- Dem Befehlsnamen folgen bei Unix-Shells durch Leerzeichen getrennte *Optionen* und *Parameter*.
- Meistens werden Optionen mit einem oder zwei einleitenden Minus-Zeichen angedeutet,
- Parameter werden hingegen einfach so angegeben.
- Welche Optionen und Parameter ein Befehl akzeptiert, muss man wissen oder nachschauen.

Beispiel

```
tantau@pcclt02:~/uebeung06> ls -l test*
-rw-r--r--  1 tantau info 547 2005-11-28 19:19 test.class
-rw-r--r--  1 tantau info 314 2005-11-28 19:19 test.ctxt
-rw-r--r--  1 tantau info 464 2005-11-28 19:19 test.java
tantau@pcclt02:~/uebeung06>
```

Wie bekommt man Hilfe?

Die Optionen `-h` oder `-help` sind oft sehr nützlich. Ebenso das Programm `man`, dem man als Parameter einen Befehlsname übergibt.

```
tantau@pcclt02:~/uebeung06> ls --help
Aufruf: /bin/ls [OPTION]... [DATEI]...
Auflistung von Informationen der DATEIen (Standardvorgabe ist das momentane
Verzeichnis). Alphabetisches Sortieren der Eintraege, falls weder -cftuSUX
noch --sort angegeben.

Erforderliche Argumente fuer lange Optionen sind auch fuer kurze erforderlich.
  -a, --all                  Eintraege, die mit . beginnen, nicht verstecken
  -A, --almost-all           implizierte . und .. nicht anzeigen
      --author               den Urheber jeder Datei ausgeben
  -b, --escape               nicht-druckbarer Zeichen oktale ausgeben
...

tantau@pcclt02:~/uebeung06> man ls
Formatiere ls(1) neu, bitte warten...
LS(1)                           User Commands                           LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...
...
```

Der aktuelle Pfad

- Während man mit einer Unix-Shell arbeitet, gibt es immer ein *aktuelles Verzeichnis*,
- das in der Regel im Prompt angezeigt wird.
- Alle *relativen Pfade* beziehen sich auf dieses Verzeichnis.

Der `pwd` (print working directory) Befehl

Der Befehl gibt das aktuelle Verzeichnis aus.

Der `cd` (change directory) Befehl

- Man gibt einen neuen Verzeichnisnamen an, dieser wird dann zum neuen aktuellen Verzeichnis.
- Insbesondere wechselt `cd ..` eine Verzeichnisebene nach oben.

Anzeigen des Inhalts eines Verzeichnisses

Der `ls` (list) Befehl

- Er zeigt eine Liste aller Dateien im aktuellen Verzeichnis an.
- Der Befehl kennt viele Optionen. Die wichtigste ist `-l`, die dafür sorgt, dass die Anzeige sehr detailliert ist.
- Als *Parameter* kann man bestimmte Dateien oder Verzeichnisse angeben. Dann werden Informationen über diese Dateien oder der Inhalt dieser Verzeichnisse angezeigt.
- Parameter können auch *Sternchen* enthalten kann. *Ein Sternchen steht immer für einen beliebigen Text.* So bezeichnet `*.pdf` alle Dateien, die auf »`.pdf`« enden.

Der `cat` (concatenate) Befehl

- Der Befehl nimmt die Namen mehrerer Dateien als Parameter und gibt deren Inhalt aneinandergereiht aus.
- Nützlich ist dieser Befehl hauptsächlich in Verbindung mit Um- und Weiterleitung der Ausgabe (später).

Der `more` Befehl

- Der Befehl zeigt eine Datei seitenweise an.
- Die verbesserte Variante dieses Befehls haben verspielte Informatiker `less` getauft.

Anzeigen des Inhalts einer Datei

Teile von Dateien betrachten

Die `head` und `tail` Befehle

- Die Befehle zeigen die ersten oder die letzten Zeilen einer Datei an.
- Mit der Option *−n*, wobei *n* eine Zahl ist, kann man angeben, wie viele Zeilen man sehen möchte.

Beispiel: `head -5 beispiel.txt`

Der `rm` (remove) Befehl

- Der Befehl löscht die Dateien, die als Parameter übergeben werden, *unwiederbringlich.*
- Er löscht jedoch *keine* Verzeichnisse und
- er löscht *nicht* rekursiv Dateien in Unterverzeichnissen.
- Mit der Option `-R` kann man allerdings erzwingen, dass Dateien doch rekursiv gelöscht werden.

Zur Diskussion

Welchen Effekt haben folgende Befehle?

```
cd /
rm -R *
```

Kopieren und Umbenennen

Der `cp` (copy) Befehl

- Der Befehl kopiert eine Datei. Parameter sind der alte und der neue Name.
- Es werden keine Verzeichnisse kopiert.

Der `mv` (move) Befehl

- Dieser Befehl verschiebt Dateien (erster bis vorletzter Parameter) an einen neuen Ort (letzter Parameter).
- Man kann den Befehl »missbrauchen«, um eine Datei umzubenennen. Dazu ist der erste Parameter der alte Dateiname und der zweite Parameter der neue Name (der neue »Ort«).

Erstellen und Löschen von Verzeichnissen

Der `mkdir` (make directory) Befehl

Der Befehl nimmt als Parameter den Namen eines neu anzulegenden Verzeichnisses.

Der `rmdir` (remove directory) Befehl

- Der Befehl löscht ein Verzeichnis,
- das leer sein muss.
- Typischerweise löscht man dazu mittels `rm *` vorher den Inhalt des Vereichnis.

Der Befehl zur Änderung von Rechten

Der `chmod` (change access mode) Befehl

- Dieser Befehl bekommt eine *Rechteänderung* und einen *Dateinamen* als Parameter.
- Die Rechteänderung besteht aus
  1. der Personengruppe, um die es geht (also `u`, `g` oder `o`),
  2. einem Minus- oder Pluszeichen (für »Recht wird entzogen« oder »Recht wird gegeben«)
  3. einem Recht (also `r`, `w` oder `x`).

Beispiel: `chmod o-r geheim.txt`

Zur Übung

```
murmel:~/temp tantau$ ls -l
total 0
-rwxr-----   1 tantau  itcs  0 Nov  1 12:32 beispiel.txt
```

Die Datei `beispiel.txt` soll »welt-lesbar« werden und das Ausführbarkeitsrecht soll gelöscht werden. Wie lauten die nötigen Befehle?

Einfach mal etwas sagen

Der `echo` Befehl

Dieser Befehl gibt einfach alle seine Parameter aus. Dieser Befehl ist hauptsächlich nützlich in der Shell-Programmierung.

Beispiel

```
murmel:~/temp tantau$ echo Hallo Welt
Hallo Welt
murmel:~/temp tantau$
```

Der `wget` (network downloader) Befehl

Dieser Befehl bekommt als Parameter eine Internetadresse (also die Adresse einer Webseite auf einem anderen Rechner). Die Datei wird geholt und eine lokale Kopie angelegt.

Beispiel

```
murmel:~/temp tantau$ wget http://www.tcs.uni-luebeck.de/index.html
--16:41:04--  http://www.tcs.uni-luebeck.de/index.html
           => 'index.html'
Resolving www.tcs.uni-luebeck.de... 141.83.63.70
Connecting to www.tcs.uni-luebeck.de[141.83.63.70]:80... connected.

...

16:41:04 (8.31 MB/s) - 'index.html' saved [9367/9367]

murmel:~/temp tantau$ head -3 index.html
<!DOCTYPE html PUBLIC "-//W3C//DTD␣HTML␣4.01␣Transitional//EN">
<html>
<head>
murmel:~/temp tantau$
```

Wörter zählen

Der `wc` (word count) Befehl

- Der Befehl bekommt einen Dateinamen als Parameter und
- zählt die Buchstaben, die Wörter und die Zeilen in der Datei.
- Mit den Optionen `-c`, `-w` und `-l` kann man entscheiden, was ausgegeben werden soll.

Beispiel

```
murmel:~/temp tantau$ cat beispiel.txt
Hallo Welt.
murmel:~/temp tantau$ wc -w beispiel.txt
       2 beispiel.txt
murmel:~/temp tantau$ wc -l beispiel.txt
       1 beispiel.txt
murmel:~/temp tantau$ wc -c beispiel.txt
      12 beispiel.txt
```

Der `diff` (difference) Befehl

- Parameter des Befehls sind zwei Dateien.
- Die Ausgabe sind alle Differenzen zwischen den Dateien in einem menschenlesbaren Format.

Der `grep` Befehl

- Die Abkürzung steht für »global search for a regular expression and print out matched lines«.
- Parameter sind ein regulärer Ausdruck und Dateinamen,
- Ausgabe sind alle Zeilen in den Dateien, in denen der reguläre Ausdruck vorkommt.
- Beispiel: `grep tantau beispiel.txt` finde alle Zeilen in `beispiel.txt`, in denen `tantau` vorkommt.

Umleitung der Ein- und Ausgabe

- Die *Ausgabe* eines Befehls wird normalerweise einfach in der Shell ausgegeben.
- Gibt man hinter einem Befehl `> dateiname` an, so wird die Ausgabe stattdessen in die angegebene Datei geleitet.
- Analog kann man mittels `< dateiname` die *Eingabe* aus einer Datei lesen lassen, statt vom Benutzer.

Beispiel

```
tantau@pcclt02:~/public_html> ls
index.html  index.html~
tantau@pcclt02:~/public_html> ls > listing
tantau@pcclt02:~/public_html> cat listing
index.html
index.html~
listing
```

Umleitung der Ein- und Ausgabe

- Man kann die *Ausgabe* eines Programmes an die *Eingabe* eines anderen Programmes leiten.
- Dazu wird eine *Pipe* benutzt, ein senkrechter Strich.

Beispiel

```
tantau@pcclt02:~/public_html> ls
index.html  index.html~
tantau@pcclt02:~/public_html> ls | wc -w
2
```

Beispiel

```
murmel:~/Documents/www/webpages/publications tantau$ grep Tantau *.bib | head -10
bibliography.bib:                  Schintke and Till Tantau and Baltasar
bibliography.bib:                  Schintke and Till Tantau and Baltasar
bibliography.bib:                  Till Tantau},
bibliography.bib:                  Till Tantau},
bibliography.bib:  author =       {Jens Gramm and Till Nierhoff and Till Tantau},
bibliography.bib:                  Tantau},
bibliography.bib:                  Tantau},
bibliography.bib:                  Tantau},
bibliography.bib:                  Tantau},
bibliography.bib:  author =       {Richard Karp and Till Nierhoff and Till Tantau},
murmel:~/Documents/www/webpages/publications tantau$
```

Was sind Shell-Skripte?

- Ein *Shell-Skript* ist eine Folge von Befehlen, die man immer wieder benutzen möchte.
- Man schreibt einfach die Folge der Befehle in eine Textdatei, jeden Befehl auf eine neue Zeile.
- Heißt die Datei beispielsweise `myscript.bash`, so kann man das Skript mittels `bash myscript.bash` aufrufen.

Beispiel

Inhalt von `myscript.bash`:

```
echo Das aktuelle Verzeichnis ist
pwd
echo Es enthaelt die folgende Anzahl an Dateien:
ls | wc -w
```

Aufruf des Skriptes:

```
murmel:~/Documents tantau$ bash myscript.bash
Das aktuelle Verzeichnis ist
/Users/tantau/Documents
Es enthaelt die folgende Anzahl an Dateien:
      16
murmel:~/Documents tantau$
```

Parameter für Shell-Skripte

- Shell-Skripte können *Parameter* bekommen.
- Innerhalb des Skriptes wird jedes Vorkommen von `$1` durch den ersten Parameter ersetzt.
- Analog wird jedes Vorkommen von `$2` durch den zweiten Parameter ersetzt und so weiter.

Beispiel

Inhalt von `backup.bash`:

```
echo Erstellung eines Backups von $1
cp $1 $1.bak
```

Aufruf des Skriptes:

```
murmel:~/temp tantau$ ls
backup.bash     wichtig.txt
murmel:~/temp tantau$ bash backup.bash wichtig.txt
Erstellung eines Backups von wichtig.txt
murmel:~/temp tantau$ ls
backup.bash     wichtig.txt     wichtig.txt.bak
murmel:~/temp tantau$ diff wichtig.txt wichtig.txt.bak
murmel:~/temp tantau$
```

Zusammenfassung

Shells

- Eine *Shell* ist ein Dialog-Programm, das die Kommunikation zwischen einem Nutzer und einer Anwendung ermöglicht.
- Eine *Unix-Shell* dient dazu, mit einem Unix-Betriebssystem zu kommunizieren.
- In dem Dialog gibt die Shell einen *Prompt* vor, der Benutzer gibt einen *Befehl* ein, die Shell zeigt *die Ausgaben des Befehlen* an und die Sache beginnt von vorne.

Wichtige Shell-Befehle

Die folgenden Shell-Befehle sollte man kennen: `cd`, `pwd`, `ls`, `cat`, `less`, `head`, `tail`, `rm`, `cp`, `mv`, `chmod`, `echo`, `wget`, `grep`, `wc` und `diff`.

Umleitung

- Schreibt man `> dateiname` hinter einen Shell-Befehl, so werden alle Ausgaben des Befehls in die Datei geschrieben.
- Schreibt man `< dateiname` hinter einen Shell-Befehl, so werden alle Eingaben des Befehls aus der Datei gelesen.
- Schreibt man `befehl1 | befehl2`, so dienen die Ausgaben von `befehl1` als Eingaben von `befehl2`.

Zusammenfassung

Shell-Skripte

- Ein Shell-Skript ist eine Datei, in der jede Zeile einen Shell-Befehl enthält.
- Man ruft ein Shell-Skript `foo.bash` auf mittels `bash foo.bash parameter1 parameter2 ...`.
- Dann werden die Zeilen des Skript nacheinander genau so ausgeführt, als hätte man sie »per Hand« nacheinander eingegeben.
- Wenn im Shell-Skript irgendwo `$1` auftaucht, so wird dieses durch den ersten Parameter ersetzt, analog mit `$2` und so weiter.