Vorlesung
Betriebssysteme und Systemsoftware

Kapitel 3:
Die Linux-Shell `bash`

Bastian Leibe

Computer Vision
Chair of Computer Science 8
RWTH Aachen University

http://www.vision.rwth-aachen.de

Viele Folien adaptiert von Klaus Wehrle und Dirk Thissen

Themenübersicht

- **Betriebssysteme und Systemsoftware**
  - Betriebssysteme: Aufbau und Aufgaben
  - Shell- und C-Programmierung
  - Prozesse und Threads, Prozessverwaltung und -kommunikation
  - CPU-Scheduling
  - Prozesssynchronisation, Deadlocks
  - Speicherverwaltung, virtueller Speicher
  - Dateisystem, Zugriffsrechte und I/O-System
  - Kommunikation, verteilte Systeme

Literatur

- **Folien zu großen Teilen orientiert an:**
  - Ehses, E.; Köhler, L.; Riemer, P.; Stenzel, H.; Victor, F.: *Betriebssysteme – Ein Lehrbuch mit Übungen zur Systemprogrammierung in UNIX/Linux*. Pearson Studium, 2005
- **Hilfreiche Informationen:**
  - Mendel Cooper: *Advanced Bash-Scripting Guide*. Rev. 10, 10.03.2014, http://tldp.org/LDP/abs/html/
  - ***Man-Pages***. Probiere: „`man man`“
- **Für Windows-Nutzer:**
  - *VitualBox: http://www.virtualbox.org*
  - *Linux-Images für VirtualBox: https://virtualboxes.org/images/*

Kapitel 3: Die Linux-Shell `bash`

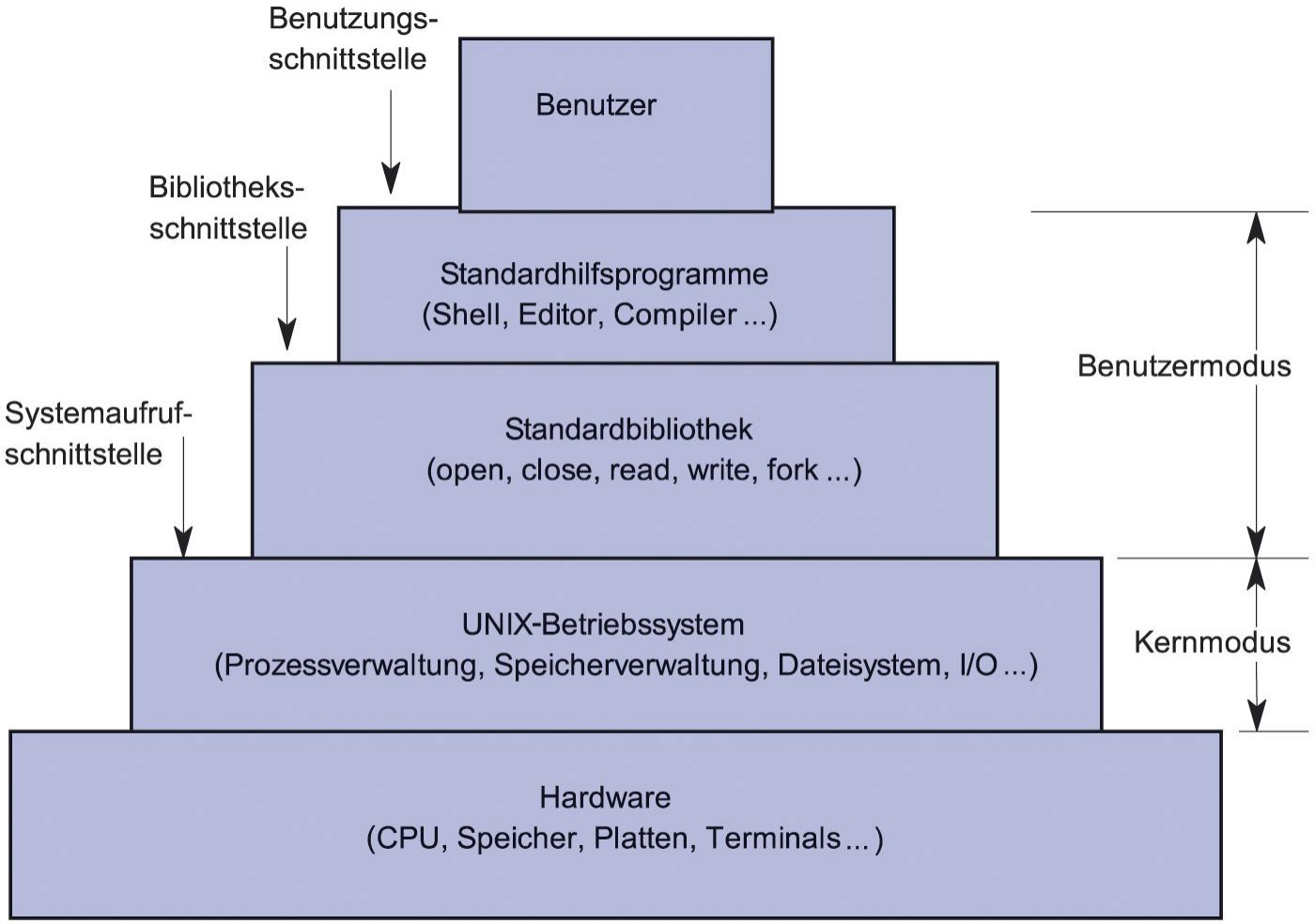
- **UNIX/Linux**
  - Aufbau, Shell
- **Grundlegende Befehle**
  - Dateiverwaltung, Pipes, Ein-/Ausgabe
- **Komplexe Shell-Kommandos**
  - Scripte, Variablen, Kommandosubstitution
- **Anweisungen und Schleifen**
  - `if`, `case`, `for`
- **Weitere hilfreiche Kommandos**

- **Entwicklung von UNIX**
  - In den 40er und 50er Jahren: Einbenutzerbetriebssysteme
  - In den 60er Jahren: Batch-Systeme (Lochkartenstapel)
  - Ab 1970:MULTICS: MULTiplexed Information and Computing Service entwickelt von M.I.T, Bell Labs und General Electrics
  - Ken Thompson (Bell Labs) entwickelt eine abgespeckte Version von MULTICS (Assembler)
  - Brian Kernighan nennt das System scherzhaft UNICS (Uniplexed Information and Computing Service)
  - UNIX
    - Thompson und Ritchie schrieben UNIX neu in C
    - UNIX Version 6 und 7 verbreitete sich schnell im akademischen Bereich

Das Betriebssystem UNIX

- IEEE entwickelt POSIX
  - Definiert API, die ein UNIX-kompatibles System unterstützen muss
- Tanenbaum, 1987: entwickelt kleinen UNIX-Klon MINIX
  - Linus Torwalds: entwickelt in Anlehnung an MINIX einen Open-Source UNIX Klon names Linux
  - Ebenfalls in C
  - Verschiedene Distributionen: Arch Linux, Debian, RedHat, Fedora, …
- Mac OS X baut auf UNIX-Kern auf
  - Ebenso später iOS für iPad, iPhone
- Android als Linux für eingeschränkte Geräte
  - Fokus auf Mobilität

- **Grober Aufbau UNIX**



- **Schnittstellen in UNIX**
  - Im Anwendungsprogramm steht die Anweisung

```
// ...
printf("Willkommen zur Vorlesung!");
// ...
```

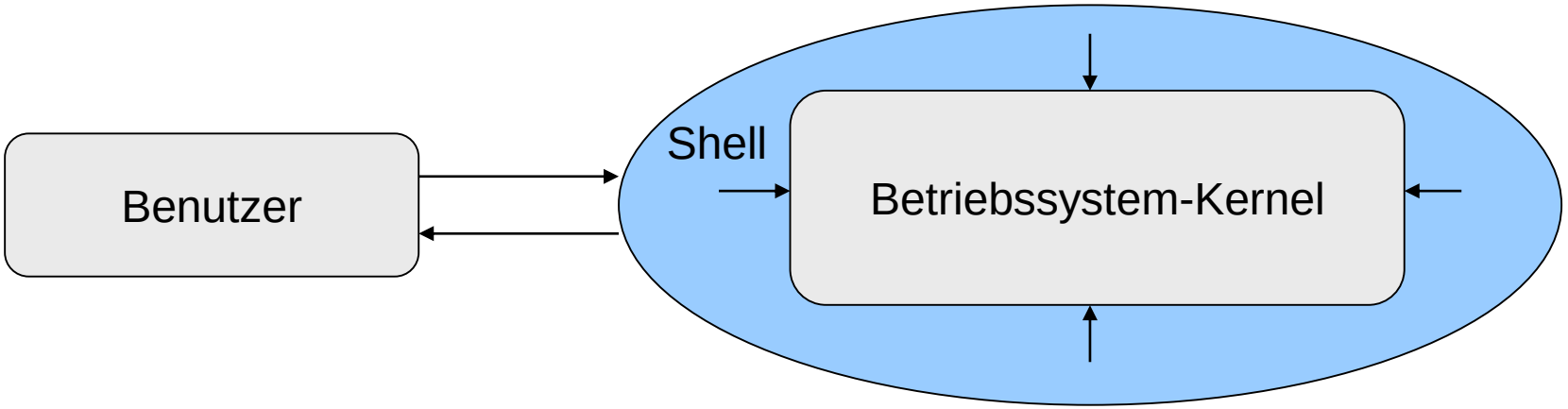
  - **`printf`** kommt aus der C Standardbibliothek (Formatierte Ausgabe)
  - Die Implementierung in der Bibliothek führt zum Aufruf der entsprechenden Systemfunktion im UNIX-Kernel:

```
write(1, "Willkommen zur Vorlesung!", 25)
```

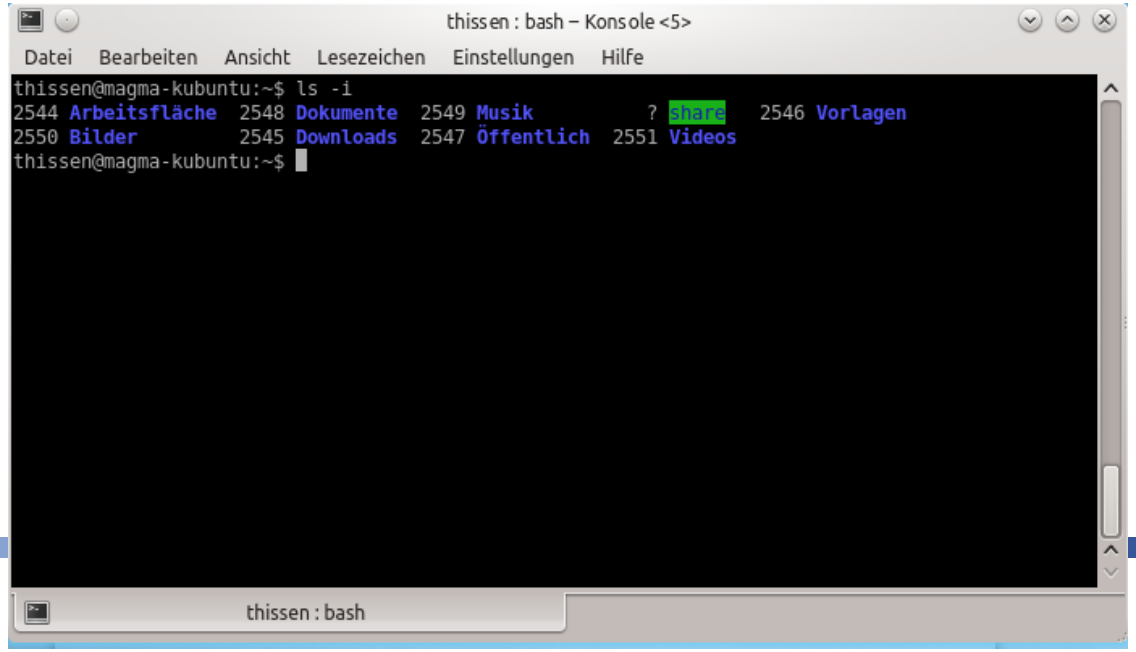
  - Diese Funktion löst einen Software-Interrupt (Trap) aus, um **`write`** im Kernel-Mode auszuführen

Was ist eine „Shell“?

- **Shell:**
  - Englisch für „Hülle“ oder „Schale“
  - Hilfsprogramm zum Starten von Anwendungsprogrammen und zum Lesen von / Schreiben auf ein Terminal
    - Verbirgt innere Details des Betriebssystems
    - Schutz des Kernels vor Benutzerzugriffen
  - Hier: Textbasierte Kommando-Prozessoren
    - `bash`, `fish`, `zsh`, `cmd.exe`, `powershell`, ...



- **Einsatzgebiete der Shell**
  - Starten von Programmen, die keine GUI anbieten
  - Problemlösung z.B. bei fehlerhaften Treibern
  - Automatisieren von Aufgaben
  - Abfrage von Informationen der Arbeitsumgebung (aktuelles Arbeitsverzeichnis, Prozess-ID des letzten gestarteten Prozesses, Uhrzeit der letzten Eingabe, Rechenzeit der letzten Anweisung, …)



Verwendung der Shell

- **Syntax einfacher Shell-Kommandos:**
  - `command argument1 argument2 ...`
  - Beispiel:
    - `echo Hallo Welt`
    - Ausgabe von „Hallo Welt“ auf dem Bildschirm

- **Shell-Script**
  - Textdatei mit Abfolge von Shell-Kommandos
  - Ähnlich wie bei höheren Programmiersprachen
  - Aufruf des Scripts: Ausführung aller Kommandos
  - Ausgabe eines Kommandos kann als Eingabe eines anderen verwendet werden

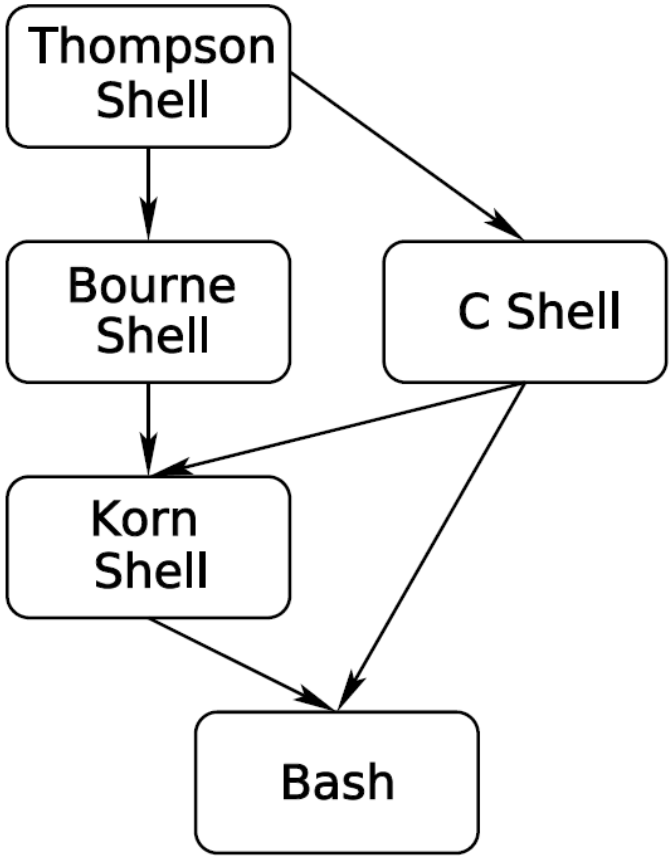
Vor- und Nachteile der Shell

- **Was spricht für Shell-Scripte?**
  - Niedrige Einstiegshürde
  - Einfache Nutzung externer Programme
  - Shell ist fast immer vorhanden (Minimalsysteme)

- **Wann sind Alternativen vorzuziehen?**
  - Anforderungen an Effizienz und Geschwindigkeit
  - Komplexere Anwendungen (Objektorientierung, Type-Checking)
  - Verwendung spezieller Datenstrukturen
  - Verwendung externe Bibliotheken
  - GUIs zur Benutzerfreundlichkeit

Verschiedene Shells

- **„Erste“ Shells**
  - 1971: Thompson-Shell
    (erste UNIX-Shell)
  - 1977/1978: Bourne-Shell (*sh*)
    (in UNIX V7)
  - 1979: C-Shell (*csh*)
    (C-ähnliche Syntax, BSD-UNIX)
  - 1988/1993: Korn Shell (*ksh*)
    (UNIX System V)
  - *1987: Bourne-Again-Shell* (***bash***)
    (Shell des GNU-Projekts, meist verbreitet:
    Linux, MAC OS X, Portierung auf fast alle UNIX-Systeme)



- **Weitere Shells: http://de.wikipedia.org/wiki/Unix-Shell**

Besondere Shells

- **Secure Shell (ssh)**
  - Ausführung einer Shell / einer Anweisung auf entferntem Rechner
  - Verschlüsselung, Authentifizierung
  - Wichtiges Werkzeug im Alltag, z.B. für github
- **Eingeschränkte Shells (restricted shell), z.B. rbash**
  - Bieten einen eingeschränkten Funktionsumfang einer normalen Shell:
    - Vorgegebene Menge von ausführbaren Anweisungen und Programmen
    - Vorgegebene Menge von zugänglichen Verzeichnissen
- **Viele Weitere: mosh, grub, busybox, …**

Grundfunktionen der Shell

- **Basisfunktionen einer Shell:**
  - Ausführen einfacher Kommandos
  - Dateinamen-Wildcards als Kommandoargumente (`rm -r *`)
  - Bedingungen (`if`, `case`) und Schleifen (`while`, `until`, `for`)
  - Interne Kommandos (`cd`, `read`)
  - Interne Variablen (`$HOME`)
  - Manipulation der Umgebungsvariablen für neue Prozesse
  - Ein-/Ausgabeumlenkung (`sort < myfile`)
  - Starten mehrerer Prozesse, Verkettung über sogenannte Pipes
  - Starten von Prozessen im Hintergrund
  - Funktionen und Funktionsaufrufe, reguläre Ausdrücke
  - Tab-Completion!

Struktur der Kommandozeile

- **Struktur der Eingabe**
  - `$ command -options arguments`

Eingabeaufforderung ("prompt") – gehört nicht zum Befehl!

Struktur der Kommandozeile

- **Struktur der Eingabe**
  - `$ command -options arguments`
- **Optionen**
  - Hier können Parameter für den Shell-Befehl eingegeben werden
  - Viele Befehle haben eine Option **`-h`** oder **`--help`**, die die verfügbaren Optionen und Eingabeformate erklärt.
- **Hilfe zu einem Befehl:** `man` **(“manual”)**
  - `$ man rm`
  - `$ man man`
  - `$ man -k delete`
    - Gibt alle Hilfe-Seiten aus, die das Schlüsselwort “delete“ enthalten

- **UNIX/Linux**
  - Aufbau, Shell
- **Grundlegende Befehle**
  - Dateiverwaltung, Pipes, Ein-/Ausgabe
- **Komplexe Shell-Kommandos**
  - Scripte, Variablen, Kommandosubstitution
- **Anweisungen und Schleifen**
  - `if`, `case`, `for`
- **Weitere hilfreiche Kommandos**

Grundlegende Befehle: `ls`

- **Inhalt eines Ordners anzeigen: `ls`** (“list”)
  - `$ ls pictures`
  - `anyFolder shell.jpg shell.pdf`
  - `$ ls -l pictures`
  -
```
drwxr-xr-x 2 leibe  prof  4096   Apr 6 21:58 anyFolder
-rw-r--r-- 1 leibe  prof  520808 Apr 6 21:42 shell.jpg
-rw-r--r-- 1 leibe  prof  570561 Apr 6 21:42 shell.pdf
```

Zugriffsrechte

- Typ: (`d`)irectory, (`l`)ink, (`-`) normale Datei
- Zugriffsrechte: Benutzer : Gruppe : andere
  (`r`)ead, (`w`)rite, e(`x`)ecute

Grundlegende Befehle: `ls`

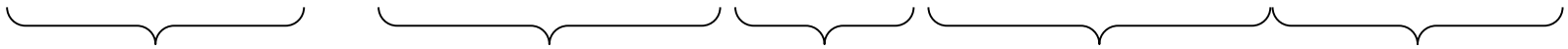
- **Inhalt eines Ordners anzeigen: `ls`** (“list”)
  - `$ ls pictures`
  - `anyFolder shell.jpg shell.pdf`
  - `$ ls -l pictures`
  -
```
drwxr-xr-x 2 leibe  prof  4096   Apr 6 21:58 anyFolder
-rw-r--r-- 1 leibe  prof  520808 Apr 6 21:42 shell.jpg
-rw-r--r-- 1 leibe  prof  570561 Apr 6 21:42 shell.pdf
```

Zugriffsrechte Besitzer

  - Benutzername: **`leibe`**
  - Gruppe: **`prof`**

Grundlegende Befehle: `ls`

- **Inhalt eines Ordners anzeigen: `ls`** (“list”)
  - `$ ls pictures`
  - `anyFolder shell.jpg shell.pdf`
  - `$ ls -l pictures`
  -
```
drwxr-xr-x 2 leibe  prof  4096   Apr 6 21:58 anyFolder
-rw-r--r-- 1 leibe  prof  520808 Apr 6 21:42 shell.jpg
-rw-r--r-- 1 leibe  prof  570561 Apr 6 21:42 shell.pdf
```



Zugriffsrechte | Besitzer | Dateigröße | Datum | Dateiname

Grundlegende Befehle: `cd`, `mkdir`, `rm`

- **Ordner wechseln: `cd`** (“change directory”)
  - `$ cd /home/leibe/`
  - `$ cd ~leibe/`
  - `$ cd ~/`

  Alle drei Befehle führen ins Home-Verzeichnis **`/home/leibe/`**
  - `$ cd ..`

  Wechsle ins Oberverzeichnis
- **Ordner anlegen: `mkdir`** (“make directory”)
  - `$ mkdir /home/leibe/bus`
- **Datei/Ordner löschen: `rm` / `rmdir`** (“remove”)
  - `$ rm /home/leibe/bus/uebungen/template.tex`
  - `$ rmdir /home/leibe/bus/uebungen/uebung01`
  - `$ rmdir -r /home/leibe/bus/`

Grundlegende Befehle: `pwd`, `cp`, `ln`

- **Aktuelles Verzeichnis: `pwd`** (“print working directory”)
  - `$ cd ~/bus/uebungen/uebung01/`
  - `$ pwd`
  - `/home/leibe/bus/uebungen/uebung01/`
- **Kopieren von Dateien: `cp`** (“copy”)
  - `$ cp ~/bus/uebungen/uebung01/uebung01.tex ./`
  - Die Datei `uebung01.tex` wird in das aktuelle Verzeichnis (`./`) kopiert
- **Erstellen von Verknüpfungen: `ln`** (“link”)
  - `$ cd ~/`
  - `$ ln -s ~/bus/uebungen/uebung01/uebung01.tex ./`
  - Dieselbe Datei ist nun in beiden Verzeichnissen verlinkt

Grundlegende Befehle: `echo`, `cat`, `wc`

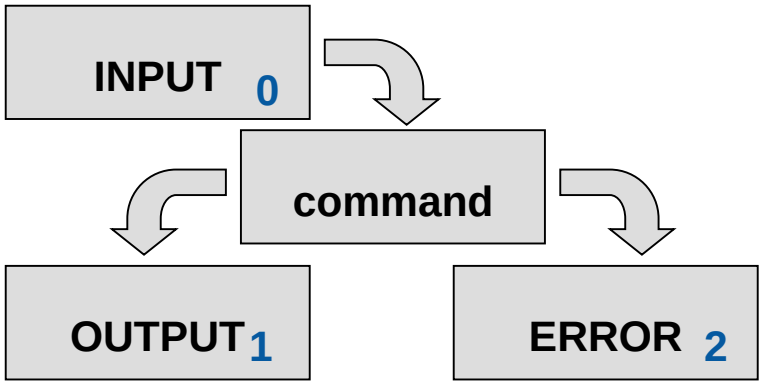
- **Text ausgeben: `echo`** (“echo”)
  - `$ echo Hallo Welt`
  - `Hallo Welt`
- **Datei ausgeben: `cat`** (“concatenate”)
  - `$ cat test.txt`
  - Der Inhalt der Datei `test.txt` wird ausgegeben.
- **Wörter/Zeilen/Zeichen zählen: `wc`** (“word count)
  - `$ wc test.txt` gibt die Anzahl der Wörter aus
  - `$ wc -l test.txt` gibt die Anzahl der Zeilen aus
  - `$ wc -c test.txt` gibt die Anzahl der Zeichen aus

Pipes

- **Einfachstes Kommando für die Shell: ausführbare Datei**
  - `$ date` → Datei `/usr/bin/date` wird ausgeführt
- **Ein Kommando endet mit Zeilenumbruch oder Semikolon:**
  - `$ cd ~/; pwd`
- **Verknüpfung von mehreren Kommandos durch eine *Pipe***
  - `$ cat test.txt | wc`
  - `$ cat test.txt; pwd | wc`
- **Will man die Ausgabe beider Kommandos mit `wc` bearbeiten, muss man klammern:**
  - `$ (cat test.txt; pwd) | wc`

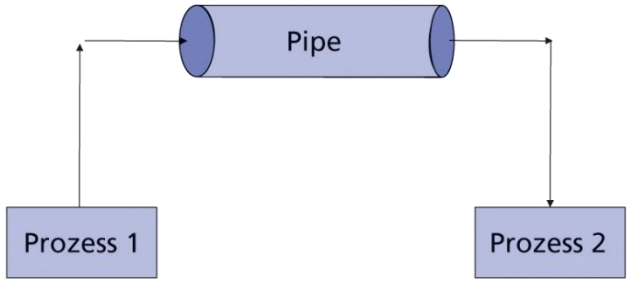
Standard-Ein- und -Ausgabe

- **Jedes Kommando verfügt über drei Kanäle:**
  - 0 – Standard-Eingabe (Tastatur)
  - 1 – Standard-Ausgabe (Bildschirm)
  - 2 – Fehlerausgabe (Bildschirm)



- **Kanäle verbinden den Befehl mit dem Betriebssystem**

- **Durch Pipes kann Kanal 1 eines Befehls in Kanal 0 des nächsten Befehls umgeleitet werden**
  - Pipes sind unidirektional



Umleiten von Ein-/Ausgaben

- **Anzapfen einer Pipe: `tee`** **(T-Stück)**
  - `$ date | tee sicherDatei | wc`
  - `$ cat sicherDatei` → Ausgabe von date
  - `$ wc < sicherDatei`
- **Ausgabe eines Programms als Eingabe eines anderen mittels „<“ und „>“:**
  - `$ wc  < temp-file  > result-file`
- **Auch: „>>“: Ausgabe wird an bestehenden Inhalt angehängt, anstatt diesen zu überschreiben**

Weitere Meta-Zeichen

- **Um einen Prozess als Hintergrundprozess auszuführen, wird ein Und-Zeichen (`&`) verwendet:**
  - `$ (sleep 10; echo 10 Sekunden um) &`
  - `3376` → Prozess-ID des Hintergrundprozesses
- **`#` dient zur Kennzeichnung eines Kommentars:**
  - `$ echo hello # world`
  - **Ausgabe: hello**
- **`*` kann als Wildcard in Parametern verwendet werden:**
  - `$ rm *` → lösche alles
  - `$ ls a*c` → suche nach Dateien, die mit a beginnen und c enden

- **UNIX/Linux**
  - Aufbau, Shell
- **Grundlegende Befehle**
  - Dateiverwaltung, Pipes, Ein-/Ausgabe
- **Komplexe Shell-Kommandos**
  - Scripte, Variablen, Kommandosubstitution
- **Anweisungen und Schleifen**
  - `if`, `case`, `for`
- **Weitere hilfreiche Kommandos**

Erzeugen eigener Kommandos

- **Wir wollen oft das folgende Kommando ausführen:**
  - `$ ls | wc -l` → Zählen der im Verzeichnis vorhandenen Dateien
- **Definiere eigenes Kommando `anz`:**
  - `$ echo 'ls | wc -l' > anz`
- **Aufruf der Shell mit `bash` und Umlenken der Eingabe → Ausführen der Datei `anz`:**
  - `$ bash < anz`
  - `$ bash anz`
  - `$ source anz`
- **Oder: Datei ausführbar machen (*Shell-Script*)**
  - `$ chmod +x anz`
  - `$ ./anz`

Parameter für Shell-Scripts

- **Bsp: Skript, um andere Dateien ausführbar zu machen**
  - Wir wollen ein Skript **`exe`** anlegen, das das folgende bewirkt:
  - **`$ exe anz`** → eine Abkürzung für: **`$ chmod +x anz`**
- **Wie muss `exe` aussehen?**

- **Parameter für Shell-Scripts (*Shell-Variablen*):**
  - `$1` wird überall im Script durch das erste Argument ersetzt,
  - `$2` durch das zweite, usw
  - Vorsicht vor Variablen die Blanks beinhalten; besser immer in doppelte Anführungszeichen setzen (`"$2"`)
- **`exe` sieht also so aus: `chmod +x "$1"`**

Parameter für Shell-Scripts

- **Gesamte Folge zur Erstellung von `exe`:**
  - `$ echo 'chmod +x "$1"' > exe`
  - `$ bash exe exe` → `exe` selbst ausführbar machen
  - `$ echo echo Es funktioniert > test` → Testprogramm
  - `$ ./exe test`
  - `$ ./test`
  - `Es funktioniert`
- **`exe` für mehrere Dateien:**
  - `chmod +x "$1" "$2" "$3"` (möglich bis `$9`)
- **`$@` bezeichnet alle Argumente :**
  - `chmod +x "$@"`
- **`$0` bezeichnet das gerade ausgeführte Programm selbst**

- **Shell-Variablen**
  - Die Zeichenfolgen **`$1, ..., $9`** sind positionelle Parameter
  - **`PATH`** ist die Liste von Verzeichnissen, in denen die Shell nach Kommandos sucht
  - **`HOME`** bezeichnet das Heimatverzeichnis des Benutzers
- **Operationen auf Shell-Variablen**
  - Wertzuweisung durch „=“
  - Lesender Zugriff mit **`$`**
  - **`$ PATH="$PATH:/home/bus/bin/"`**

Shell-Variablen

- Definition eigener Variablen
  - `$ dir='/home/bus'`
  - `$ cd "$dir"`
- Auch möglich: interaktive Variablenbelegung durch `read`:
  - `$ read -p "Bitte Name eingeben: " name`
  - `$ echo Dein Name ist "$name"`

Shell-Variablen

- **Anzeigen der Werte aller Shell-Variablen mittels `set`:**
  - `$ set`
  - `HOME = ...`
  - `PATH = ...`
  - `...`
  - `dir = ...`
- **Variablen werden standardmäßig nicht an aufgerufene Programme vererbt!**
  - Auch nicht an weitere Instanzen der Shell
- **Aber: exportieren möglich mit `export`**
  - `export` befördert Shell-Variablen zu Umgebungsvariablen

Kommandosubstitution

- **Kommandosubstitution heißt ein Verfahren, bei dem ein Kommando zum Teil oder komplett aus einer Ausgabe besteht**
- **Beispiel:**
  - `echo Im Verzeichnis existieren "$(ls | wc -l)" Einträge`
  - Bildschirmausgabe:
    „Im Verzeichnis existieren 15 Einträge“
  - In Java:
    `System.out.println("Im Verzeichnis existieren " + countWords(listDir()) + " Einträge");`
  - Statt `$(...)` wird oft auch `` `...` `` benutzt, was aber nicht schachtelbar ist. (Vorsicht: `` ` `` ist nicht `'`)

- **UNIX/Linux**
  - Aufbau, Shell
- **Grundlegende Befehle**
  - Dateiverwaltung, Pipes, Ein-/Ausgabe
- **Komplexe Shell-Kommandos**
  - Scripte, Variablen, Kommandosubstitution
- **Anweisungen und Schleifen**
  - `if, case, for`
- **Weitere hilfreiche Kommandos**

If-Anweisung

- **Durch Verwendung eines `if` sind Verzweigungen möglich:**
  - `if <anweisung>; then`
  - `  <anweisungen>`
  - `else` → der `else`-Teil kann auch entfallen
  - `  <anweisungen>`
  - `fi`
- **Jede Anweisung liefert einen exit-Code:**
  - 0 erfolgreich
  - Wert ungleich 0 nicht erfolgreich
- **Beispiele:**
  - `[ -r pfad ]; echo $?` existiert **`pfad`** und ist lesbar?
  - `[ -d pfad ]; echo $?` existiert **`pfad`** und ist ein Directory?

If-Anweisung

- Beispiel zu `if`:
  - `$ v1="Ich"`
  - `$ v2="Du"`
  - `if [ "${v1}" = "${v2}" ]; then`
  - `  echo Ich bin Du`
  - `else`
  - `  echo Irgendwas ist hier falsch`
  - `fi`
- Die <anweisung> im „`if`“ kann auch ein Vergleich mittels `[ … ]` sein.

- **Mögliche Vergleichsoperationen:**
  - Strings:
    - `Str1 = Str2` sind Strings gleich?
    - `Str1 != Str2` sind Strings nicht gleich?
  - Ganze Zahlen:
    - `-eq` gleich
    - `-ne` nicht gleich
    - `-gt` größer als
    - `-ge` größer gleich
    - `-lt` kleiner als
    - `-le` kleiner gleich

Case-Anweisung

- **Syntax der `case`-Anweisung:**
  - ```
    case <wort> in
    <muster>)
      <anweisungen>
      ;;
    <muster>)
      <anweisungen>
      ;;
    <...>
    esac
    ```
- **`*)` als Muster für Default**
    ```

Schleifen: For

- **For-Schleife**
  - `for <variable> in <Liste von Worten>`
    `do`
    `  <command>`
    `done`
  - Durchläuft jedes Element der Liste und führt `<command>` aus
- **Beispiel: Schleife, die in drei Zeilen Worte ausgibt:**
  - ```
    for i in "Dies ist" ein Test
    do
      echo "$i"
    done
    ```
    ```

Schleifen: For

- Alternative:

```
for((i=0;i<4;i++));
do
  echo "$i"
done
```

Weitere Schleifen-Anweisungen

- **`while`-Anweisung:**
  - **`while`** `<condition>`
    `do`
    `<anweisungen>` → ausgeführt, solange `<condition>` wahr
    `done`

- **`until`-Anweisung:**
  - **`until`** `<condition>`
    `do`
    `<anweisungen>` → ausgeführt, solange `<condition>` falsch
    `done`

- **UNIX/Linux**
  - Aufbau, Shell
- **Grundlegende Befehle**
  - Dateiverwaltung, Pipes, Ein-/Ausgabe
- **Komplexe Shell-Kommandos**
  - Scripte, Variablen, Kommandosubstitution
- **Anweisungen und Schleifen**
  - `if`, `case`, `for`
- **Weitere hilfreiche Kommandos**

- **`grep` (global regex print) – suchen von Textmustern**
  - `$ grep [Optionen] Muster Datei1 ... [DateiN]`
  - Gibt Zeilen aus, die einem Suchmuster entsprechen
- **`tr` (truncate) – löschen und ersetzen von Zeichen**
  - `$ echo Test | tr -d 'T'` → liefert „`est`“ als Ausgabe
  - Entfernt oder ersetzt alle Vorkommen eines Musters in einer Eingabe
- **`head` – Ausgabe des ersten Teils einer Datei**
  - `$ head -5 datei` → liefert die ersten 5 Zeilen von „`datei`“
- **`tail` – Ausgabe des letzten Teils einer Datei**
  - `$ tail -5 datei` → liefert die letzten 5 Zeilen von „`datei`“

- **sed (stream editor) – automatisierte Manipulation per Kommandozeile**
- **Liest und transformiert Eingabe zeilenweise**
- **Mächtige Syntax**
  - Ersetzen von Ausdrücken:
    ```
    $ sed 's/alt/neu/g' Eingabedatei > Ausgabedatei
    ```
  - Ersetzen mehrfacher Leerzeichen durch ein einziges:
    ```
    $ sed 's/ \+/ /g' Eingabedatei > Ausgabedatei
    ```
  - Auch: Entfernen von Zeilen, die ein Muster enthalten:
    ```
    $ sed '/muster/d' Eingabedatei > Ausgabedatei
    ```
  - Benutzung mit Pipes:
    ```
    $ ls | sed '/[0-9]\+/d' | wc -l
    ```
  - Viele weitere Einsatzmöglichkeiten

Dateiübertragung

- **`nc` (netcat)**
  - **`cat`**: schreibt Dateien in die Standardausgabe (und kann auch anderes)
  - **`nc`**: schreibt Dateien auf ein *Netzwerk*
- **Beispiel:**
  - **`nc mail.server.net 25`**
    Öffne eine Verbindung zu mail.server.net auf Port 25
  - Auch als Server zu verwenden:
    `nc -l -p 12345`
  - Datenübertragung:
    - Erste Konsole: **`$ cat datei | nc -l -p 12345`**
    - Zweite Konsole: **`$ nc 127.0.0.1 12345 > file`**

Weitere Kommandos

bash	Die Bourne Again Shell
cat	Konkatenieren und Ausgeben von Dateien auf die Standardausgabe
chmod	Verändern des Schutzmodus für Dateien
cmp	Vergleich von Dateien auf Gleichheit
cp	Kopieren einer Datei
cut	Erzeugt für jede Spalte eines Dokumentes eine eigene Datei
date	Gibt Datum und Uhrzeit aus
diff	Gibt alle Unterschiede zwischen zwei Dateien aus
echo	Ausgabe seiner Argumente
find	Aufsuchen aller Dateien, die eine gegebene Bedingung erfüllen
grep	Durchsuchen einer Datei nach Zeilen mit einem vorgegebenen Muster
kill	Senden eines Signals an einen Prozess
ln	Erzeugen eines Links auf eine Datei
ls	Auflisten der Dateien eines Verzeichnisses
make	Recompilieren der veränderten Teile eines großen Programms
mkdir	Erzeugen eines Verzeichnisses
mv	Bewegen oder Umbenennen einer Datei
paste	Kombination mehrerer Dateien als Spalten einer Datei
pwd	Ausgabe des aktuellen Arbeitsverzeichnisses
rm	Löschen einer Datei
rmdir	Löschen eines Verzeichnisses
sleep	Suspendierung der Ausführung für eine gegebene Zeit (in Sek.)
sort	Sortieren einer Datei aus ASCII- Zeilen
wc	Zählen von Zeichen, Wörtern und Zeilen einer Datei

Terminals

- **Man kann in `bash` durchaus `bash&` aufrufen…**
  - Wer malt die Buchstaben auf den Monitor?
- **Früher wurde I/O von einem Terminal ausgegeben**
- **Heutzutage benutzen wir dafür Terminal Emulatoren**
- **Häufig Unterstützung für Signale, Farben, Maus, …**
- **Erlaubt die Erstellung von TUIs**
- **Zum Beispiel, `tmux` (terminal multiplexer)**
  - **Emuliert Terminals in einem (emulierten) Terminal**
  - **Erlaubt Terminal Sessions die länger sind als eine Verbindung**
  - **Kann mehrere Terminals in einem darstellen**

Zusammenfassung

- **Shell-Kommandos und –Scripte**
  - Einfache Shell-Kommandos erlauben schnelle Lösung kleiner Aufgaben zur Rechnerverwaltung
  - Scripte sind mächtige Hilfsmittel
  - Immer abzuwägen: Reicht ein Shell-Script, oder sollte man die Aufgabe besser in einer Hochsprache lösen?
    - **z.B.** `python` ist auch interpretiert und oft ebenfalls verfügbar

- **Das passende Werkzeug macht die Aufgabe leichter!**

Themenübersicht

- **Betriebssysteme und Systemsoftware**
  - Betriebssysteme: Aufbau und Aufgaben
  - Shell- und C-Programmierung
  - Prozesse und Threads, Prozessverwaltung und -kommunikation
  - CPU-Scheduling
  - Prozesssynchronisation, Deadlocks
  - Speicherverwaltung, virtueller Speicher
  - Dateisystem, Zugriffsrechte und I/O-System
  - Kommunikation, verteilte Systeme