Grundlagen der Informatik für Ingenieure I

Background:

1 Einführung in UNIX

1.1 Was ist ein Betriebssystem?

- **Ein Betriebssystem**

  - **steuert** die gesamten Hardwarekomponenten eines Rechensystems; wobei das Gesamtsystem ein verteiltes System (= mehrere vernetzte Rechner) sein kann.

  - **ermöglicht** durch die Bereitstellung einer Programmierschnittstelle (Betriebssystemschnittstelle; Systemcalls) dem Benutzer eine abstrakte Sicht auf das System (keine Hardwaredetails).

1.1 Was ist ein Betriebssystem?

Ein Betriebssystem

◆ verfügt in der Regel über keine Information aus den Applikationen, die auf ihm abgewickelt werden.

◆ Trotzdem erwartet man:

- faire Abwicklungsstrategien auf der Basis einer vorgegebenen Kostenfunktion (z. B. Timesharing, Realtime, Datenbanksystem, Num. Langläufer),
- eine Betriebsmittelbereitstellung *just-in-time* für die gerade laufende Applikation (Speicher, Plattenspeicher, E/A-Geräte),
- und die Vermittlung des Gefühls, dass man als Benutzer über seine eigene Maschine verfügt, wenn mehrere Benutzer am gleichen System arbeiten.

1.2 Begriffsbildung

(Die einzelnen Begriffe werden in späteren Kapiteln vertieft behandelt!)

◆ **Single-User-Systeme:**

- Nur ein Benutzer kann zu einem Zeitpunkt eine Maschine nutzen.
- Dies war in den 50er - Jahren im Allgemeinen der Fall;
- heute findet man dies meist noch auf PCs.

◆ **Multi-User-Systeme:**

- Viele Benutzer können ein System gleichzeitig benutzen;
- sie teilen sich die meist große Rechenleistung des Systems,
- jeder Benutzer (besser: Prozess) bekommt zeitscheibenweise (im 100ms-Bereich) den Prozessor zugeteilt (Timesharing-Systeme).
- Nimmt man nur Teilleistungen eines Rechners in Anspruch, spricht man von Servern (Dateiserver, DB-Server, Web-Server, Nummerische Hochleistungsserver, etc.).

1.2 Begriffsbildung

◆ **Multiprocessing:**

- In einem Multiuser-System werden mehrere “**Benutzerumgebungen**” parallel verwaltet.
- Aktiv heißt, der Prozessor arbeitet den Code (das Programm) einer Benutzerumgebung ab.
- Man nennt diese Abarbeitung einen **Prozess**.
- Bei einem Ein-Prozessor-System kann jeweils nur ein **Prozess** aktiv sein.
- Bei einem Mehrprozessorsystem können entsprechend mehrere **Prozesse** gleichzeitig aktiv sein (--> Multiprocessing)
- Ein Benutzer kann auch mehrere Prozesse (für verschiedene Aufgaben) besitzen.

1.2 Begriffsbildung

1.2 Begriffsbildung

◆ **Verteilte/Parallele Systeme:**

- Eine Applikation kann auch durch mehre Prozesse realisiert sein.
- Diese Prozesse können auf **gemeinsame Daten** zugreifen oder durch **Nachrichtenmechanismen** Daten austauschen.
- Diese Verarbeitungsart nennt man bei einem Prozessor **quasi parallel**, auf mehreren Prozessoren **echt parallel.**
- Bei einem **Verteilten System** sind die Prozessoren räumlich verteilt.

1.2 Begriffsbildung

◆ **Multi-Threading-Systeme**

- Ein **Prozess** kann selbst wiederum eine Anzahl von Aktivitätsträgern enthalten, die in der gleichen Umgebung ablaufen.
- Diese Aktivitätsträger nennt man **Threads**.
- Diese können wiederum je nach Hardwarevoraussetzung (Ein- oder Mehrprozessorsysteme) quasi parallel oder echt parallel ablaufen.

◆ Unixsysteme sind potentiell multiuser-, multi processing und multithreading-Systeme, Mit Unixsystemen sind Verteilte Systeme realisierbar, wobei einzelne wiederum Multiprozessorfähig sein können.

◆ Die Betriebssystemschnittstelle (API=Application Programmining Interface) ist international standardisiert. Betriebssysteme bieten heute diese Schnittstelle (wenigstens als Untermenge) an.

1.3 UNIX - Historische Entwicklung

- 1969 erste Implementierung auf einer PDP7 (Ken Thompson)
- 1971 Reimplementierung in der Sprache B
- 1973 Erste Version in C
- 1976 UNIX Version 6
- 1978 UNIX Version 7 - erste portable Systemversion
- ab 1980 Entstehung der Berkeley-Linie
  1981 4.1BSD
  1983 4.2 BSD
  1986 4.3 BSD
  1989 4.4 BSD
- 1982 erste kommerziell vertriebene UNIX-Version von AT&T (System III)
- 1984 UNIX SystemV / Release 2
- 1986 UNIX SystemV / Release 3
- 1989 UNIX SystemV / Release 4
- Heute: Posix-Standard
  Einige bekannte Systeme: Solaris, HP-UX, AIX, Linux (SUSE, RedHead, Debian), diverse Varianten von Ectzeitsystemen für eingebettete Systeme

1.4 Vorbemerkung

- Soweit Sie bereits mit Rechnern gearbeitet haben, ist Ihnen sicher die Steuerung des Rechners (ausschließlich) über graphische Oberflächen geläufig. Die Zielsetzung solcher Oberflächen ist, den Benutzer die Notwendigkeit von Hintergrundwissen zu “ersparen”. Das ist natürlich nicht die Idee dieser Vorlesung! Wir möchten Ihnen in Zusammenhang mit der Benutzung von Rechnern auch vermitteln
  - wie Interaktionen mit Hardware und Betriebssystem organisiert sind,
  - welche Basismechanismen vorhanden sind, mit deren Hilfe Gesamtsysteme konstruiert werden können, zu welchem Zweck auch immer,

  und zwar sowohl aus Benutzersicht, als auch aus der Sicht von Programmsystemen.
- Aus diesem Grunde werden wir - mindestens anfänglich - direkt mit einem sog. Kommandoprozessor (Shell) arbeiten, der als Kommandos nur Zeichensequenzen *versteht*. Tatsächlich handelt es sich um eine spezielle Sprache, die der Kommandoprozessor interpretiert und die gewünschte Funktionalität mit Hilfe von Betriebssystembasisfunktionen realisiert.
  (Es gibt auch Programme bestehend aus Kommandos, sog. Shell-Scripts)
- Die grafische Oberfläche mit der Sie im CIP-Pool arbeiten werden heißt “KDE”, V 3.3.

1.5 Login, Benutzerumgebung

- UNIX unterscheidet zwischen Groß- und Kleinschreibung *(casesensitiv)*!!!

- **Benutzeranmeldung über Loginfenster:**
  - **Login name: *accountname*** z. B. siuwlins <tab> oder <r>
    - Standardaufbau eines *Accountnamens*:
      - s für Student,
      - i für Ingenieurwissenschaften (allgemein für die Fakultät)
      - die ersten beiden Buchstaben des Vornamens,
      - aufgefüllt mit den ersten Buchstaben des Nachnamens auf maximal 8 Zeichen;
      - bei entstehenden gleichen ***Accountnamen*** wird “variiert”.)

    `Password:` <r> oder Klick auf GO-Button

- **Abmelden:**
  - Öffnen eines Menüs mit rechter Mousetaste auf Bildschirmhintergrund; dort "Abmelden" selektieren!

1.5 Login, Benutzerumgebung

- **Benutzerumgebung** (Standard im CIP-Pool: KDE):
  - die voreingestellte Basis-Benutzerumgebung umfasst folgende Punkte:
    - **Benutzername**
    - **Identifikation** (User-Id und Group-Id)
    - **Home-Directory** (dort werden Ihre Dateien und Directories abgelegt)
    - **Shell** (Kommandointerpreter)
  - die notwendigen Daten sind in der Passwortdatei zusammengefasst Form eines Eintrags:

```
Benutzername : verschl. Paßwort : uid : gid : Kommentar : Home-Dir : Shell
```

  - Beispiel:

```
siuwlins:MsnbnQMyHhsvw:4002:4000:Uwe Linster:/u0/siuwlins:/bin/tcsh
```


1.6 Sonderzeichen

- einige Sonderzeichen haben unter UNIX besondere Bedeutung
  - Funktionen:
    - Korrektur von Tippfehlern
    - Einwirkung auf den Ablauf von Programmen
  - Übersicht:

<BACKSPACE>	letztes Zeichen löschen (häufig auch <DELETE>)
<tab>	sh versucht automatisch Katalog- (directory) oder Dateinamen zu ergänzen
<CTRL>U	alle Zeichen der Zeile löschen
<CTRL>C	Programm wird abgebrochen - Interrupt
<CTRL>D	End-of-File

1.7 Dateisystem

- **Hierarchisches Dateisystem** in dem Dateien der folgenden Arten existieren:

  ◆ **reguläre Dateien**
  - Bytefolge ohne Struktur
  - kann sowohl Text, als auch Binär-Information enthalten
  - wenn man diese Bytefolgen liest oder schreibt, spricht man von ***"i/o-streams"***

  ◆ **spezielle Dateien** *(special files)*
  - verweisen i. a. auf Peripheriegeräte - Integration physikalischer Geräte in das Dateisystem! ***(/dev)***

  ◆ **symbolic links**
  - verweisen auf andere Dateien oder Kataloge

1.7 Dateisystem

◆ **Kataloge *(Directories)*** mit folgenden möglichen Einträgen:

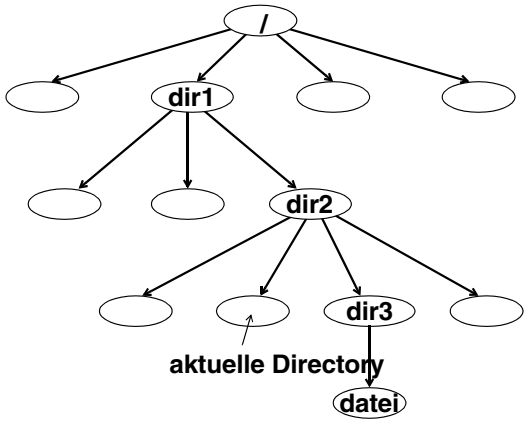
- Dateinamen und Katalognamen (Katalog ist spezielle Datei) und Verweise auf die Dateien *(inode)*
  Dateiname max. 255 Zeichen
- Katalogname **.** verweist auf den Katalog selbst
- Katalogname **..** verweist auf den Vorgängerkatalog

1.7 Dateisystem

- die Wurzel *(root)* des Dateisystems ist mit *"/"* benannt.
- ein Pfadname wird aus mehreren ***Directory***-Namen und evtl. einem Dateinamen zusammengesetzt:
  - absolut: von der root beginnend oder
  - relativ: vom "current directory" aus gesehen.
- Trennsymbol in Pfaden ist ebenfalls *"/"*

  z. B. /dir1/dir2/dir3/datei
  oder ../dir3/datei

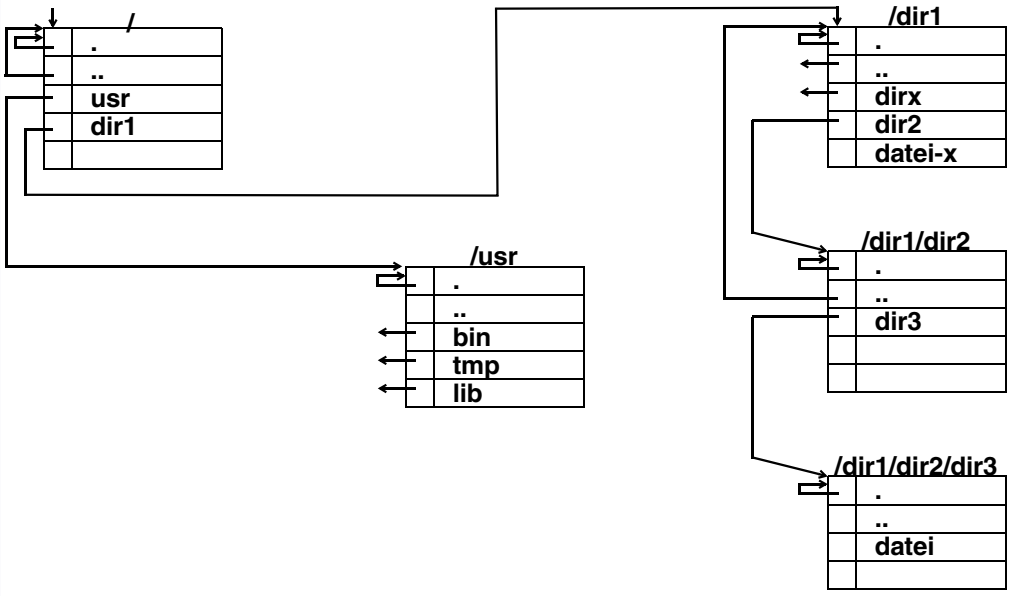
1.7 Dateisystem



- durch das Kommando **cd** kann die aktuelle ***Directory*** gewechselt werden
  z. B. `cd ../dir3`

1.7 Dateisystem

- Andere Darstellung der ***Directory*** Struktur



1.7 Dateisystem

- **Zugriffsrechte:**
  - Jede Datei ist mit individuellen Zugriffsrechten versehen, die für
    - den **Eigentümer** der Datei ***(user),***
    - für alle Benutzer aus der **Gruppe**, zu der die Datei gehört ***(group)*** und
    - für alle übrigen Benutzer ***(others)***

    getrennt gesetzt werden können.

  - Zugriffsrechte werden unterteilt in:

    r Leserecht

    w Schreibrecht

    x Ausführungsrecht

1.7 Dateisystem

◆ Zugriffsrechte können vom Eigentümer einer Datei mit Hilfe des Kommandos ***chmod(1)*** verändert werden.

◆ Beispiel: Directory mit 4 Dateien

- -rw------- 1 bolch i4staff 1213 Feb 20 12:41 MyRect.class
- -rw-r--r-- 1 bolch i4staff 741 Feb 20 12:41 MyRect.java
- -rw-rw-r-- 1 bolch i4staff 988 Feb 20 12:41 TestMyRect.class
- -rw-rw-rw- 1 bolch i4staff 770 Feb 20 12:41 TestMyRect.java

1.8 UNIX-Kommandointerpreter - Shell

■ **Kommandosprache**:

- Benutzer kommuniziert unter der Verwendung einer Kommandosprache mit dem Betriebssystem UNIX
- Der **Kommandointerpreter** (**Shell**) überprüft Eingabe und leitet dann das Kommando an UNIX weiter.

■ Auf den meisten UNIX-Rechnern stehen mehrere **Shells** zur Verfügung:

- **sh** **Bourne-Shell** - erster UNIX-Kommandointerpreter
- **csh** **C-Shell** (vor allem für interaktive Benutzung geeignet)
- **tcsh** **erweiterte C-Shell** - enthält zusätzliche Editier-Funktionen, wie ***vi*** oder ***xemacs*** (allgemein Standard im CIP-Pool)

1 Aufbau eines UNIX-Kommandos

- **UNIX-Kommandos** bestehen aus:
  - **Kommandonamen:**
    - Der Name einer Datei in der ein **ausführbares Programm** oder eine **Kommandoprozedur** (= mehrere Kommandos) für die Shell abgelegt ist (*x-permission*).
  - einer Reihe von **Optionen** und **Argumenten:**
    - Kommandoname, Optionen und Argumente werden durch Leerzeichen oder Tabulatoren voneinander getrennt.
    - Optionen sind meist einzelne Zeichen, denen ein *"–"* vorangestellt ist.
    - Argumente sind häufig Namen von Dateien, die von dem Kommando bearbeitet werden.
    - Beispiele:

      `ls -l` (Auflisten von Dateien eines Directories)

      `cp paula.java dummy.java` (Kopieren von Dateien)

      `cp -r * ..` (Kopieren aller Dateien und Directories nach ..)

2 Vordergrund- / Hintergrundprozess

- Eine Shell meldet mit einem **Promptsymbol,**
  `Rechnername [Pfadname] >,`
  `z. B.: faui02t [~/GdI/Aufgaben/Aufgabe1] >`
  dass sie ein Kommando entgegennehmen kann.

- Die Beendigung des Kommandos wird abgewartet, bevor ein neues Promptsymbol ausgegeben wird: **Vordergrundprozess.**

- Wird am Ende eines Kommandos ein **&**-Zeichen angehängt, erscheint sofort ein neues Promptsymbol - das Kommando wird im Hintergrund bearbeitet: **Hintergrundprozess.**
  - Beispiel:

    `xemacs` `HelloWorld.java` **`&`**

3 E/A-Kanäle eines Kommandos

◆ Jedes Kommando (Programm) wird beim Aufruf mit 3 E/A-Kanälen versehen:

stdin	Standard-Eingabe (Vorbelegung = Tastatur),
stdout	Standard-Ausgabe (Vorbelegung = Terminal) und
stderr	Fehler-Ausgabe (Vorbelegung = Terminal).

◆ Diese E/A-Kanäle können auf Dateien umgeleitet werden oder auch mit denen anderer Kommandos verknüpft werden ***(Pipes).***

3 E/A-Kanäle eines Kommandos

◆ Umlenkung der E/A-Kanäle auf Dateien:

- Die Standard-E/A-Kanäle eines Programms können von der Shell aus umgeleitet werden.
  (z. B. auf Dateien oder auf andere Terminals)
- Die Umleitung eines E/A-Kanals erfolgt durch die Zeichen *“<“* und *“>”*, gefolgt von einem Dateinamen.
- Durch *“>”* wird die Datei ab Dateianfang überschrieben; wird statt dessen *“>>”* verwendet, wird die Ausgabe an die Datei angehängt.

3 E/A-Kanäle eines Kommandos

- Syntax-Übersicht:

< datei1	legt den Standard-Eingabekanal auf **datei1**, d. h. das Kommando liest von dort (statt von der Tastatur)
> datei2	legt den Standard-Ausgabekanal auf **datei2** (statt aufs Terminal)
> & datei3	*(csh, tcsh)* legt Standard- und Fehler-Ausgabe auf **datei3.**
2 > datei4	*(sh, ksh)* legt den Fehler-Ausgabekanal auf **datei4.**

3 E/A-Kanäle eines Kommandos

- Durch eine **Pipe** kann der Standard-Ausgabekanal eines Programms (Kommandos) mit dem Eingabekanal eines anderen verknüpft werden
- Die Kommandos für beide Programme werden hintereinander angegeben und durch *"|"* getrennt.
- Beispiel:

```
ls -al | wc
```

- Das Kommando **wc** (Wörter zählen), liest die Ausgabe des Kommandos **ls** und gibt die Anzahl der Wörter (Zeichen und Zeilen) aus.

4 Kommandoausgabe als Argumente

◆ Die Standard-Ausgabe eines Kommandos kann einem anderen Kommando als Argument gegeben werden, wenn der Kommandoaufruf durch ``` geklammert wird.

Beispiel:

```
rm `grep -l XXX *`
```

- Das Kommando **`grep -l XXX`** liefert die Namen aller Dateien, die die Zeichenkette **XXX** enthalten auf seinem Standard-Ausgabekanal.

  ➥ Es werden alle Dateien gelöscht, die die Zeichenkette **XXX** enthalten - **also Vorsicht: keine Warnung, kein *UnDo*!!!!**

5 Quoting

◆ Wenn eines der Zeichen mit Sonderbedeutung (wie **<**, **>**, **&**) als Argument an das aufzurufende Programm übergeben werden muss, gibt es folgende Möglichkeiten dem Zeichen seine Sonderbedeutung zu nehmen:

- Voranstellen von \ nimmt genau einem Zeichen die Sonderbedeutung
  \ selbst wird durch \\ eingegeben.

- Klammern des gesamten Arguments durch **" "**,
  " selbst wird durch **\"** angegeben.

- Klammern des gesamten Arguments durch **' '**,
  ' selbst wird durch **\'** angegeben.

6 Environment

(Nur zur Info - nichts ändern!!!)

- Das *Environment* eines Benutzers (Benutzerumgebung) besteht aus einer Reihe von ***Text-Variablen***, die alle abgefragt werden können.
- Mit den Kommandos ***env(1)*** bzw. ***printenv(1)*** können die Werte der Environment-Variablen angezeigt werden:

Environment

```
% env
HOME=/home/cip/guest/uelinste
LOGNAME=uelinste
MANPATH=/man:/usr/man
PATH=/local/bin:/usr/ucb:/bin:/usr/bin:.
SHELL=/local/bin/tcsh
TERM=term
USER=uelinste
HOST=faui06a
```


6 Environment

(Nur zur Info - nichts ändern!!!)

- Auf Environment-Variablen kann – wie auf normale Shell-Variablen auch – durch ***$Variablenname*** in Kommandos zugegriffen werden.
- Mit dem Kommando ***setenv(1)*** (C-Shell) bzw. ***set*** (Shell) können Environment-Variablen verändert und neu erzeugt werden:

```
setenv PATH $HOME/bin:$PATH

set PATH=$HOME/bin:$PATH;
```

- Mit dem Kommando ***echo $Variablenname*** kann die Environment-Variable abgefragt werden

6 Environment

(Nur zur Info - nichts ändern!!!)

◆ Überblick über einige wichtige Environment-Variablen:

USER	Benutzername (BSD)
LOGNAME	Benutzername (SystemV)
HOME	Homedirectory
TTY	Dateiname des Login-Geräts (Bildschirm)
TERM	Terminaltyp (für bildschirmorientierte Programme, z. B. ***vi(1)***)
PATH	Liste von Directories, in denen nach Kommandos gesucht wird
MANPATH	Liste von Directories, in denen nach Manual-Seiten gesucht wird (für Kommando ***man(1)***)
SHELL	Dateiname des Kommandointerpreters (wird teilweise verwendet, wenn aus Programmen heraus eine Shell gestartet wird)

1.9 Auswahl einiger Kommandos

ls	Directory auflisten
wichtige Optionen für ls:	
-l	langes Ausgabeformat
-a	auch mit **”.”** beginnende Dateien werden aufgeführt
cat	Datei(en) hintereinander ausgeben
more, less	Dateien bildschirmweise ausgeben
cp	Datei(en) kopieren
mv	Datei(en) verschieben oder umbenennen
rm	Datei(en) löschen
mkdir	Directory erzeugen
rmdir	Directory löschen (muss leer sein!!!)

1.9 Auswahl einiger Kommandos

lpr	Datei ausdrucken
chmod	Zugriffsrechte einer Datei verändern
wc	Zeilen, Wörter und Zeichen zählen
grep, fgrep	nach bestimmten Mustern bzw. Zeichenketten suchen
sort	sortieren
who, finger	Liste der gerade am Rechner aktiven Benutzer
rwho	wie who, aber über alle am Netz angeschlossenen Rechner
slogin	login auf einem anderen Rechner

1.10 Editoren

xemacs

- ***xemacs*** ist ein sehr mächtiger Editor — benötigt allerdings auch sehr viel Speicherplatz.
  - Er enthält eine Anbindung an X-Windows.
  - Er enthält einen Lisp-Interpreter zur Bearbeitung von Makros.
  - Er enthält Makropakete zur komfortablen Interaktion mit C-Compiler, Java, HTML, Debugger, ftp, mail, ...
  - Er unterstützt durch Syntax-Prüfung die Programmstrukturierung

- Aufruf

```
xemacs datei.standardextension &
```

1.10 Editoren

- Ein wichtige Anmerkung:
  - Die Bearbeitung der Texte durch Editoren geschieht in der Regel in Datenpuffern, die im Arbeitsspeicher gehalten werden.
  - Erst durch die Ausführung eines ***"save"*** oder ***"save buffer"*** wird der aktuelle Inhalt auf eine Datei herausgeschrieben.
  - Sollten Sie z. B. immer wieder die gleichen Fehlermeldungen von einem Compiler bekommen, obwohl Sie ständig den Programmtext ändern, so liegt es häufig daran, dass Sie vergessen haben, den Pufferinhalt auf die Datei herauszuschreiben, der Compiler also immer noch auf die “unveränderten” Daten auf der Platte zugreift!

1.11 Anhang

- ***homedirectory****:* userspezifische “Wurzel”-*directory;* $HOME
- ***currentdirectory****:* Die *directory* in der Sie sich gerade befinden.
- Dateiname = Pfadname
  vollständiger Name durch den Dateibaum
  - von der *root-dir* aus: mit “/” beginnend
  - relativ vom *currentdir*ectory aus
- “Bewegen” des *currentdir-pointers:*
  - cd - *change directory*
- Wo bin ich?
  - pwd - *print working directory*
- Auflisten einer *directory*
  - ls; ls -l; ls -al - *list directory*

1.11 Anhang

◆ Zugriffsrechte:

	user	group	world
	rwx	rwx	rwx

◆ Vorbelegung: rw- r-- r--

◆ Ändern der Zugriffsrechte
- chmod

◆ Anlegen einer *directory*
- mkdir *directoryname*

◆ Wiederholen des letzten Kommandos (sehr wichtig!!)
- ↑

◆ Mit der **Tabulatortaste** können Dateinamen und Kommandos vervollständigt werden (auch sehr wichtig!!)

1.11 Anhang

◆ Standard-I/O am Beispiel *cat - concatenate*

Kommando	**Eingabe**	**Ausgabe**
cat filename1	filename1	*stdout*
cat fn1 fn2 > fn3	fn1 fn2	fn3 *(*= neue Datei*)*
cat fn1 fn2 >> fn3	fn1 fn2	fn3 (= exist. Datei)
cat	*stdin*	*stdout*
cat > fn4	*stdin*	fn4

◆ Kommandos zur Darstellung von Textdateien (auf dem Screen)

more filename

less filename

head filename

tail filename

1.11 Anhang

- Pipe
  - ls -l | wc
  - cat fn1 | wc > fn5
- *pattern matching*
  - grep ‘Hello’ HelloWorld.java
    - sucht nach der Zeichenfolge: Hello (in der Datei HelloWorld.java)
  - fgrep ‘Hello’ HelloWorld.java
    - sucht nach dem Wort: Hello (in der Datei HelloWorld.java)