Computergrundlagen
Einführung in UNIX

Maria Fyta

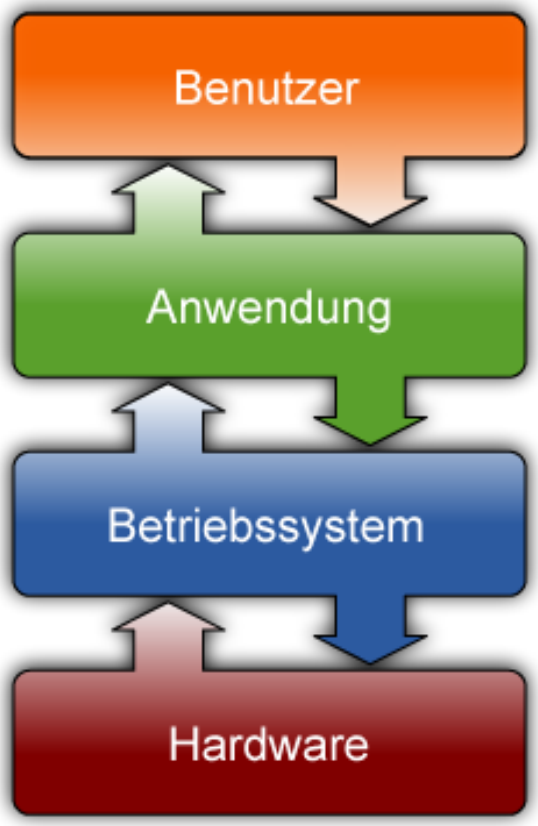
Institut für Computerphysik
Universität Stuttgart

Wintersemester 2012/13

Überblick

- Was ist ein Betriebssystem?
- Architektur von Betriebssystemen (Unix, Linux) und Geschichte
- Unix Systeme - Ein(-aus)loggen - Kennwort
- Unix - Befehle
- Unix Dateisystem
- Verzeichnisse und Dateien
- Links
- Prozesse - Pipelines
- Eingabe und Ausgabe
- Root-Konto
- Betriebsystem-Shell
- Textverarbeitungsprogramme (editors) (vi, emacs)
- Graphikverarbeitungsprogramme (gimp,xmgrace)
- Versionsverwaltung von Dateien (CVS, SVN, Git)

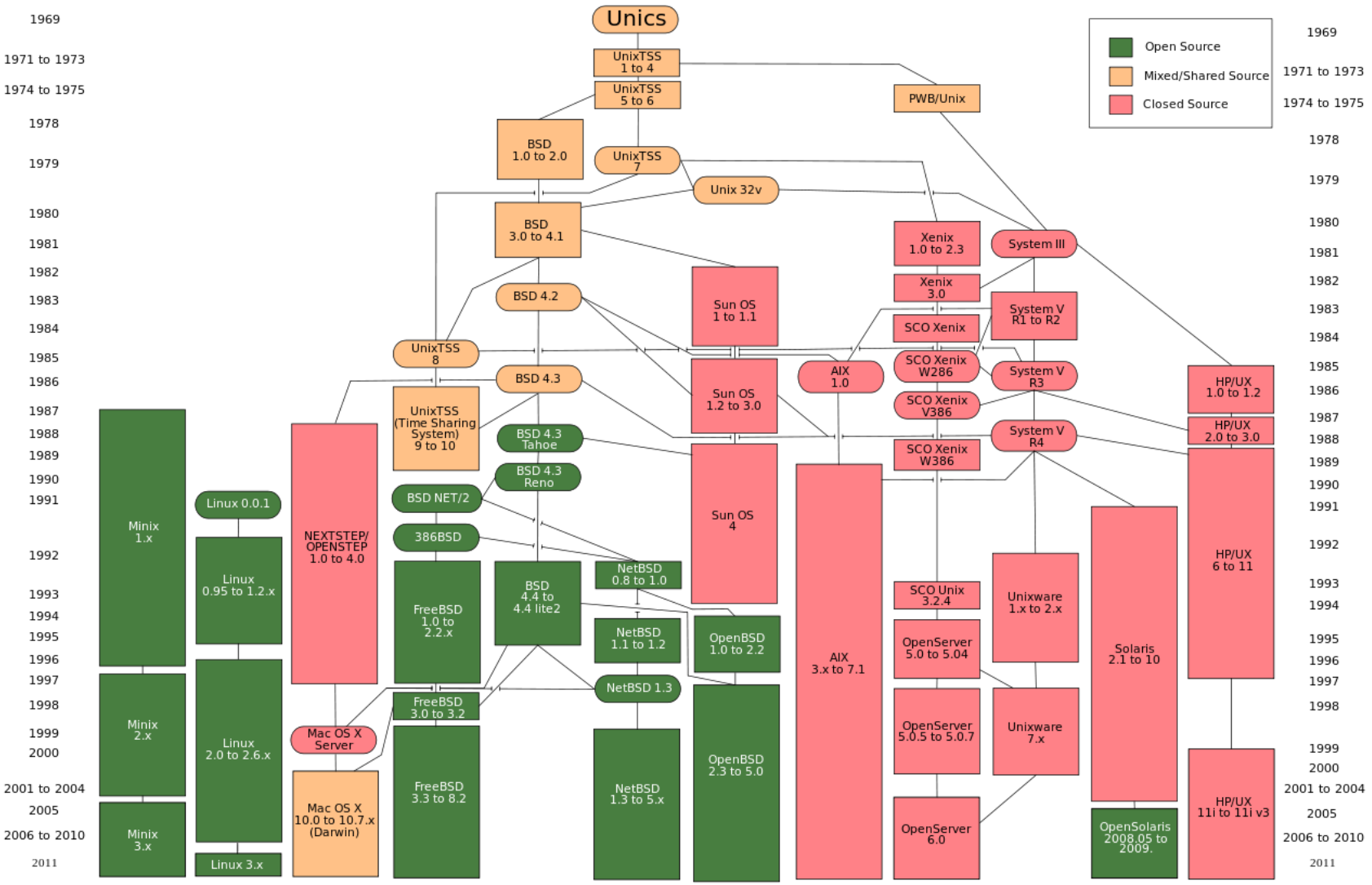
Betriebssysteme



- Vermittler zwischen Benutzer, Programmen und Hardware
- Philosophie der Benutzerschnittstelle
- meist *nicht* graphisch (DOS, UNIX, Cisco IOS)

Kurze Geschichte des Unix

Mehrbenutzersystem (multi-user), Mehrprozessbetrieb (multi-tasking environment), Stabilität, Portabilität



Betriebssysteme

- Windows: dominiert PC-Markt
- UNICES: (Linux), Mac OS X, IBM AIX, Oracle Solaris, ...
- Supercomputer Top-500 von 2010

Betriebssystem	Installationen	GFlops
Linux	455	27.162.011
other UNICES	23	1.702.295
Windows	5	412.590
andere	17	3.257.787

- UNIX-Systeme dominieren wissenschaftliches Rechnen
- Großrechner am HLRS nutzen verschiedene UNICES
- ... und auch die meisten Internet-Server

Warum?

Windows	Linux & andere UNICES
GUI integraler Teil des Systems	Terminal und/oder X11 + Desktop
Software erwartet oft Admin-Rechte	Benutzer ohne Admin-Rechte, extra root-Account
meist lokale Anwendungen	Anwendungen netzwerk-transparent
graphische Administration	per Terminal administrierbar
fernwartbar mit kommerzieller Software	out-of-the-box fernwartbar
Mein Computer	*Unser* Computer

Und woher bekomme ich Linux?

- es gibt nicht ein, sondern viele Linuxe — Distributionen
- Live-CDs: ausprobieren ohne Installation
- Dual-Boot: Installation parallel mit anderen OS
- Achtung: manchmal lässt Windows keinen Platz mehr — dann löschen die Installer es nach *einer* Nachfrage!

Distributionen

- (K)Ubuntu - benutzerfreundlich, mit Gnome/Unity bzw. KDE-GUI
  `http://www.ubuntu.com`, `http://www.kubuntu.org`
- Xubuntu: einfacher, aber auch schneller, für Netbooks
  `http://www.xubuntu.org`
- OpenSuSE: sehr benutzerfreundlich, einfache Administration
  `http://www.opensuse.org`

UNIX-Grundlagen

- mehrere Programme laufen gleichzeitig: Prozesse
- *ein* Programm für *eine* Aufgabe
- Komplexität durch Verknüpfen von Prozessen
- *alles* wird durch Dateien repräsentiert
- es gibt *einen* Verzeichnisbaum
- Benutzer und Gruppen haben Rechte an diesen Dateien

Architektur des Linux Betriebsystemes (I)

- Kernel
  Umfasst Gerätettreiber (Grafikkarte, Netzwerkkarte, Festplatten, usw.), Speicherverwaltung, Unterstützung für verschiedene Dateisysteme.
  Den Kernel findet man in /boot/vmlinuz (binäre Form) und in /usr/src/linux (Quelldatei)
- Shells und GUIs
  Linux unterstützt 2 verschiedene Befehleingaben:
  1. zeichenorientierte Befehlszeilen (command lines) (z.B. sh oder bash - Bourne shell, csh - C shell)
  2. graphische Benutzeroberfläche (graphical interface-GUIs), z.B. KDE oder GNOME window manager.

  (Fernzugang meistens durch eine wörtliche Befehlszeile - command line).

Architektur des Linux Betriebsystemes (II)

- Dienstprogramme (system utilities)
  - Fast alle Unix Dienstprogramme sind auf Linux portiert (z.B. Befehle wie ls, cp, grep, awk, sed, bc, wc, more, usw.). Diese haben sehr spezifische Aufgaben, wie z.B.:
    - grep findet eine Zeichenkette in eine Datei,
    - wc zählt die Anzahl der Wörter, Zeilen und bytes in eine Datei)

    Diese Befehle können einfach kombiniert werden statt ein monolithisches Applikationsprogram zu schreiben.

  - Die Linux Programme enthalten auch viele nützliche Server Programme, die daemons. Diese unterstützen Fern-Netzwerk und Verwaltungsdienstleistungen (z.B. telnetd und sshd bieten Fern-Einloggen, lpd Druckerdienstleistungen, httpd dient Webseiten, usw.). Ein daemon wird beim Systemstart automatisch hervorgebracht und "wartet" bis ein Ereignis auftrettet.

Architektur des Linux Betriebsystemes (III)

- Anwendungen
  - Die Linux Distributionen enthalten typischerweise viele standard nützliche Applikationsprogramme. Beispiele sind:
    - emacs (Dateiaufbereiter-editor),
    - xv (Bildbetrachter), gcc/g++ (C/C++ Compiler),
    - xfig (Zeichenpaket),
    - LaTeX(leistungsstarke Formatierungssprache) und
    - soffice (StarOffice ein MS-Office Klon das Word, Excel und Powerpoint Datein öffnen und schreiben kann).
  - Redhat Linux kommt auch mit rpm, den Redhat Package Manager, mit dem man sehr einfach Applikationen installieren und deinstallieren kann.

Unix Systeme: Ein- und Ausloggen (I)

1. Zeichenorientierte Terminale (TTY):

   Anmeldung mit telnet oder lokal. Man bekommt das Prompt:
   `login:`

   - Hier gibt man den Benutzername und drückt die Taste enter/return $\hookleftarrow$. (Unix ist schreibungsabhängig). Dann benötigt das System das Kennwort:
     `login:mfyta`
     `password:`
   - Man tippt das Kennwort, dann wieder die Taste enter/return $\hookleftarrow$. Das Kennwort wird natürlich nicht angezeigt. Wenn man sich vertippt, fängt man wieder von vorne an. Ansonsten, bekommt man ein Shellsymbol:
     `$`
     Mit "exit" oder "logout" oder Ctrl-D kann man sich aus dem zeichenoriertierten Unix shell ausloggen.

Unix Systeme: Ein- und Ausloggen (II)

2. Graphische Terminale

   Hier bekommt man ein grafisches Prompt für Benutzername und Kennwort. Wenn man sich einlogged kann man einen grafischen window manager sehen, ähnlich wie bei Microsoft Windows. Dort gibt es Menüs oder Iconen füer "shell", "xterm", "console", oder "terminal emulator". So wechselt man wieder zu ein shell Prompt. Durch den Menü Optionen "Log out" oder "Exit" kann man sich wieder ausloggen.

- Tipp: Das Kennwort ändern:

  Der Unix Befehl ist :
  `$ passwd` ↩
  Das System benötigt das alte Kennwort und danach das neue, das man 2 mal eintippen muss (um Fehler zu verhindern). Es ist wichtig ein sicheres Kennwort zu finden (keine Wörteraud dem Wörterbuch, mindestens 7-8 Zeichen lang, gemischte Zahlen, Buchstaben und Satzzeichen. Keine Zeichen benutzen die man nicht auf allen Tastaturen finden kann. Das Kennwort muss man geheim halten.)

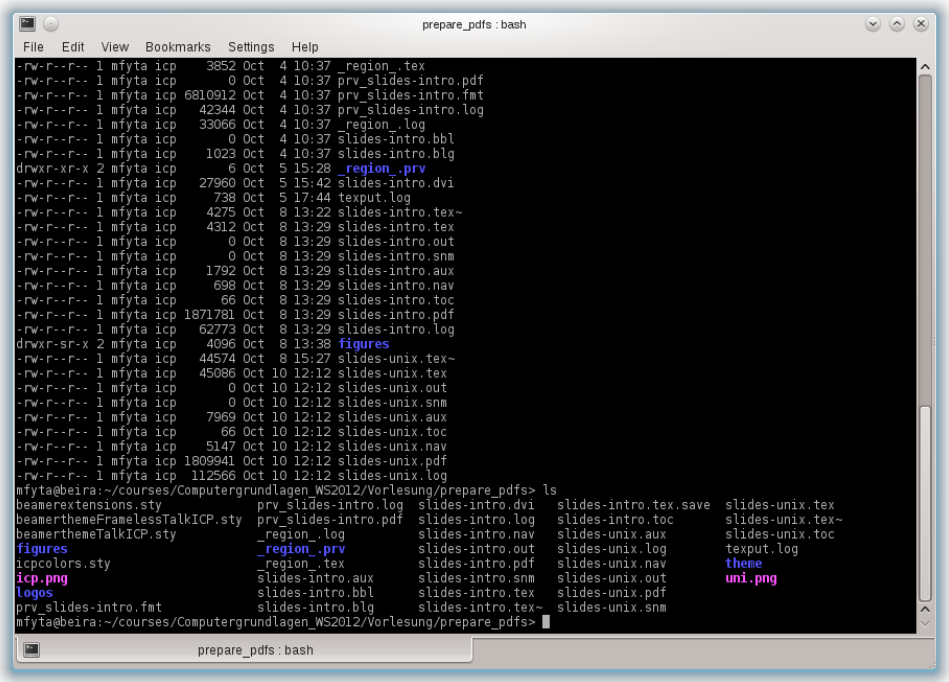
Allgemeines Format der Unix Befehle

Eine Unix Befehlzeile besteht aus dem Namen eines Unix-Befehles (der Befehl ist der Name eines eingebauten -built-in- shell Befehles, ein System utility oder eines Anwendungsprogrammes) durch seine "Argumente" (Optionen und die Ziel-Dateinamen und/oder Ausdrücken). Die allgemeine Syntax dür einen Unix Befehl ist:

```
$ command -options targets ↩
```

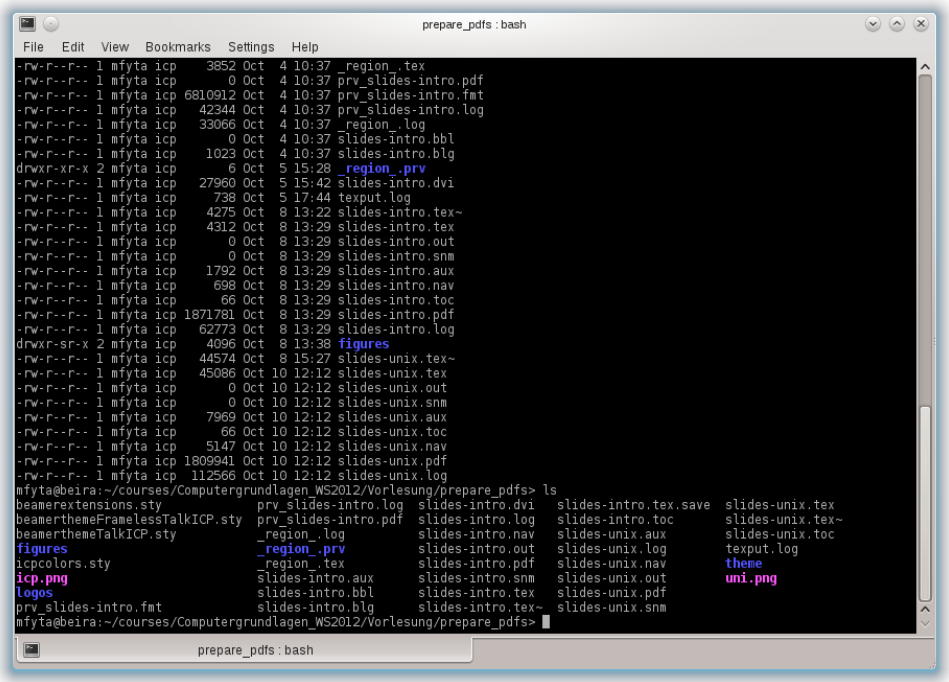
Hier "command"(Befehl) darf ein Verb, "options" (Optionen) ein Adverb und "targets"(Ziele) die direkt Objekte eines Verbes sein. In dem Fall, der Benutzer mehrere Optionen eingeben will, müssen diese nicht immer separat aufgefürt werden. Die Optionen können auch alle gemeinsam nach einem Halbgeviertstrich aufgelistet werden.

Interface: Terminal und Shell



- *Shell* in einem *Terminal*
- *Terminal*: Tastatur-Eingabe und Zeichen-Ausgabe
- *Shell*: startet und verwaltet Programme
- funktioniert genauso auch per Netzwerk

Grundlegende Shell-Benutzung



- Cursor-Up/Down: vorherige Befehle wiederholen
- Tabulator: automatische Ergänzung von Dateinamen
- Zeilen sind editierbar
- linke Maustaste: markieren, mittlere: einfügen
- Control-a/e: Anfang/Ende der Zeile
- Genaueres hängt vom Shell-Typ ab (sh, bash, csh,...)

Grundlegende Dateisystembefehle

`man <program>`	Hilfe, verlassen mit q
`info <program>`	Ausführliche Hilfe
`ls <file>...`	Datei(en) auflisten
`cp [-r] <src>... <dst>`	Dateien kopieren (-r: rekursiv)
`mv <src>... <dst>`	Dateien verschieben/umbenennen
`rm [-r] <file>...`	Dateien löschen
`pwd`	aktuelles Verzeichnis ausgeben
`mkdir <dir>...`	Verzeichnis erzeugen
`cd <dir>...`	das Arbeitsverzeichnis wechseln
`cat <file>...`	Textdatei auf Terminal ausgeben
`less/more <file>...`	Textdatei seitenweise anschauen
`[u]mount <dev> <dir>...`	Ein-/aushängen von Laufwerken

Dateien

- alle Daten werden in *Dateien* gespeichert
- Dateien können in *Verzeichnissen* zusammengefasst werden
- eine Datei wird durch Ihren *Pfad* identifiziert
- besondere Pfade:

`.`	aktuelles Verzeichnis
`..`	Übergeordnetes Elternverzeichnis
`~`	eigenes Benutzerverzeichnis (Home)
`~name`	Benutzerverzeichnis (Home) des Benutzers `name`

- Dateien, die mit „.“ beginnen, sind versteckt
- Die Shell (und jedes andere Programm) hat ein aktuelles Arbeitsverzeichnis
- Pfade, die nicht mit „ / “ oder „ ~ “ anfangen, sind *relativ* zum Arbeitsverzeichnis

Dateien finden (I)

Mindestens 3 Unix Befehle mit den man Dateien auffinden kann.

1. `find`: Wenn man eine ungefähre Vorstellung der Verzeichnisstruktur hat

   `$ find` Verzeichnis `-name` Zieldatei `-print`

   Der Befehl sucht die Zieldatei überall unter dem gegebenen Verzeichnis.

2. `which`: Wenn man ein Anwendungsprogramm durch eine Eingabe des Names im Shell ausführen kann, kann man diesen Befehl benutzen um herauszufinden wo dieses auf der Festplatte gespeichert ist:

   ```
   $ which ls
   /bin/ls
   ```

3. `locate`: ist sehr schneller als `find` wenn man Dateien sucht deren Namen einen bestimmten Zuchbegriff haben oder wenn man in einem grossen Dateiraum sucht (z.B. unter \).

   `$ locate “.txt”`

   Der Befehl findet alle Dateinamen im Dateisystem die “.txt” irgendwo in deren vollen Pfade enthalten.

Dateien finden (II)

Noch einige Beispiele

- `$ find /home -name "*.txt" -print 2>/dev/null`
  sucht in allen Verzeichnissen Dateien die in ".txt" enden und gibt die passenden Dateien mit deren Pfad aus. Die Ausführungszeichen (`) sind nötig um Ausbreitung der Dateinamen zu vermeiden. `2>/dev/null/` verdrängt Fehlermeldungen (z.B. wenn das Inhalt von Verzeichnissen für die der Benutzer keine Rechte hat nicht lesbar ist).

- `$ find . -name "*.txt" -exec wc -l '' ';'`
  zählt die Anzahl der Zeilen in jeder Textdatei in und unterhalb des aktuellen Verzeichnisses. '{}'wird durch den Namen jeder gefundene Datei erzetzt und ';' schliesst die Ausführungsklausel (`-exec`).

Dateien finden (III)

→ `find` kann auch Dateien nach Typ (z.B. `-type f` für Dateien, `-type d` für Verzeichnisse), nach Berechtigungen (z.B. `-perm o=r` für alle Dateien und Verzeichnissen die by anderen gelesen werden kann, nach Größe (`-size`) finden.

→ `locate` speichert alle Dateinamen des Systemes in einem Index der nur einmal am Tag aktualisiert wird. Dies bedeutet dass der Befehl keine Datein finden kann die kürzlich erstellt wurden. Anderseits findet er Datein die vor kurzem gelöscht wurden.

Text in Dateien finden - `grep`

→ `grep` (General Regular Expression Print)

- `$ grep` Optionen Vorlage Dateien
  Der Befehl sucht die angegebenen Dateien (oder die Standard Eingabe, wenn keine Dateien bennant werden) für Zeilen die zu eine bestimmte Vorlage passen.
  z.B. `$ grep hello *.txt`
  durchsucht alle Text-Dateien im aktuellen Verzeichnis für Zeilen die "hello" enthalten.
  Einige nutzbare Optionen:
  `-c`: zält die Anzahl der entsprechenden Zeilen,
  `-i`: Groß-/Kleinschreibung ingorieren,
  `-v` die Zeile die nicht passen ausdrücken,
  `-n` Ausdruck der Zeilenummer vor der entsprechende Zeile.

Weiter mit grep-Beispiele

```
$ grep -vi hello *.txt
```

sucht durch alle Dateien im aktuellen Verzeichnis für Zeilen die keine form des Wortes “hello” (z.B. Hello, HELLO, or hELlO).

Falls man Dateien in einem gesamten Verzeichnisbaum füer ein bestimmtes Muster suchen möchte, kann man `grep` mit `find` combinieren. Dazu benutzt man einfache Anführungszeichen um die Ausgabe von `find` in `grep` zu übergeben.

```
$ grep hello ‘find . -name “*.txt“  -print‘
```

Dieser Befehl durchsucht alle Text-Dateien in der Verzeichnisstruktur innerhalb des aktuellen Verzeichnisses für Zeilen die das Wort “hello” enthalten.

Weiter mit grep - Details (I)

Die Mustern die `grep` benutzt, sind als reguläre Ausdrücke bekannt. Sowie numerische Ausdrücke werden reguläre Ausdrücke aus grundlegenden Teilausdrücke durch Operatoren kombiniert.

Der wichtigste Ausdruck ist ein regulärer Ausdruck der mit ein einzelnes Zeichen übereinstimmt. Die meisten Zeichen, einschließlich aller Buchstaben und Ziffern sind reguläre Ausdrücke die mit sich selbst übereinstimmen. Jedes anderes Zeichen mit besonderer Bedeutung kann durch "\" angegeben werden.

Weiter mit grep - Details (II)

Eine Liste von '[' und ']' stimmt mit jedes einzelnes Zeichen in der Liste zu. Wenn "^" das erste Zeichen der Liste ist, dann passt es auf jedes Zeichen das nicht in der Liste ist. Man kann eine Reihe von Zeichen eingeben durch einen Bindestrich ("-") zwischen dem ersten und letzten Element der Liste. In diesem Sinne, [0-9] passt zu jeder Ziffer und [^a-z] jedes Zeichen das keine Ziffer ist. Die Zeichen '^' und '$' sind spezielle Zeichen die den Anfang bzw. das Ende einer Zeile bestimmen. Das Zeichen '.' passt mit jedem Zeichen.

$ grep^..[l-z] hello.txt

Der Befehl entspricht jede Zeile in hello.txt, die eine dreimälige Reihenfolge enthält. Diese Zeichenfolge ended mit einem Kleinbuchstaben von l bis z.

Suchen und Finden: Zusammenfassung

grep: Suchen in Dateien

`grep [-ri] <string> <file>...`

- sucht nach String oder Muster in Textdateien
- `-r`: auch in Unterverzeichnissen
- `-i`: Groß-/Kleinschreibung ignorieren
- wer mehr braucht: `awk`

find: Suchen von Dateien

`find <path>... <expression>`

- Suchen von Dateien und Verzeichnissen
- `find . -size +500c`

  $\Longrightarrow$ sucht Dateien mit $>$500 Zeichen
- `find . -name "*.txta -mtime -1`

  $\Longrightarrow$ sucht .txt-Dateien, die vor $<$24h geändert wurden

Das Unix Dateisystem I

Umfasst Kernel, Programmdatei für alle Befehle, Information zur Konfiegurierung, Benutzer Datei, spezial Dateien zur Hardware und Betriebssystem Bedienung. Alles was in das Unix Dateisystem gespeichert ist gehört in eins von den 4 Typen:

1. Normale Dateien (ordinary files)
2. Verzeichnis (directory)
3. Einheiten (devices)
4. Links

Das Unix Dateisystem II

1. Normale Dateien (ordinary files)

   Diese enthalten Text, Dateien oder Programminformationen und dürfen nicht andere Dateien oder Verzeichnisse enthalten. Im Gegensatz zu anderen Betriebssystemen werden Unix-Dateisysteme nicht in einem Namen-Teil und einem Fortsatz geteilt. Alle Zeichen der Tastatur (ausser '') bis zu 256 Zeichen lang können benutzt werden. Zeichen wie $\star$,?,#, \ haben besondere Bedeutung in den Shells und müssen mit Vorsicht genommen werden. Dateinamen mit Leerzeichen sind nicht so einfach zu manipulieren - lieber den Unterstrich _ benutzen.

2. Verzeichnis (directory)

   Diese enthalten Ordner, Dateien und sonstige Verzeichnisse.

Das Unix Dateisystem III

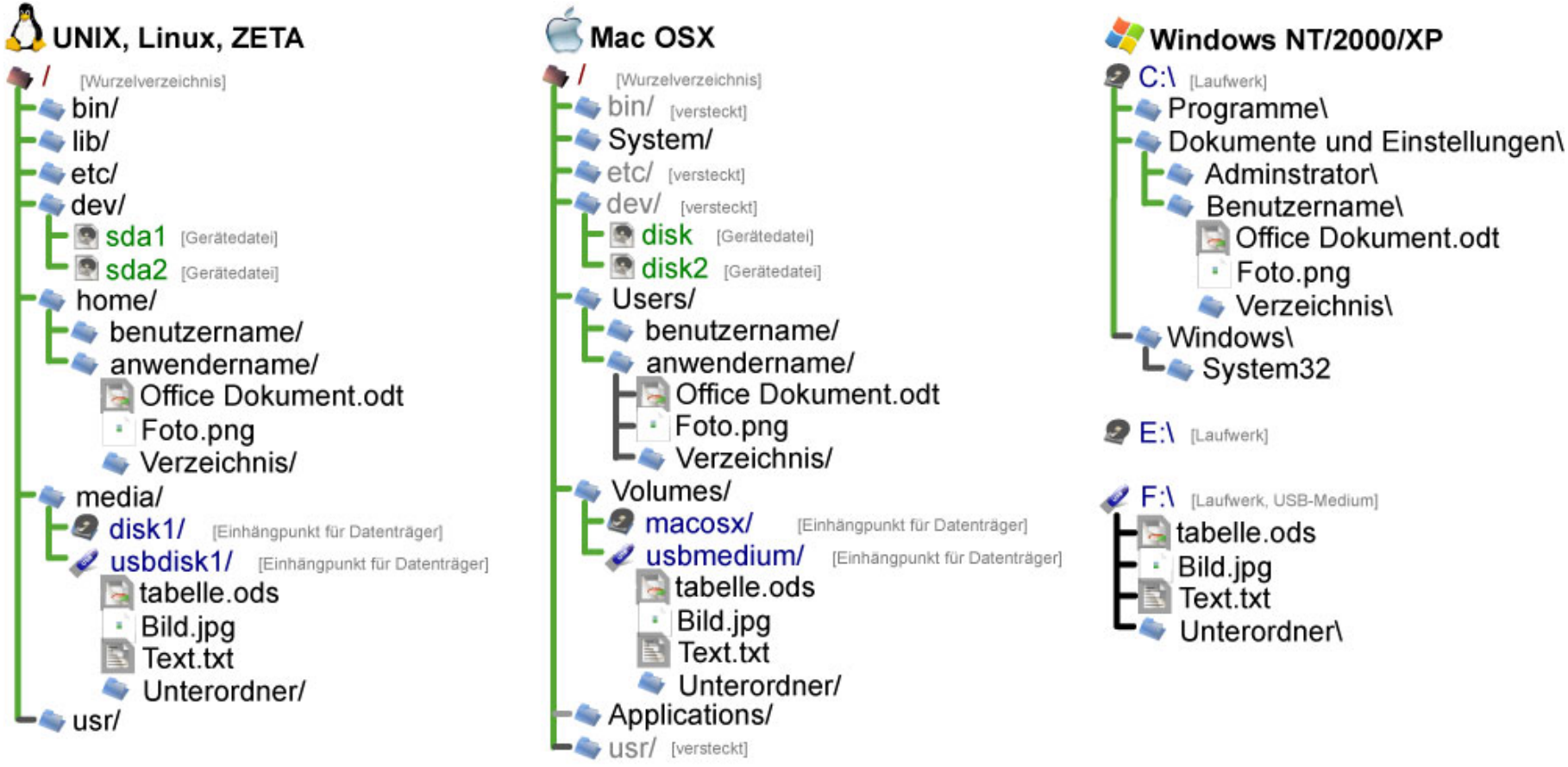
3. Einheiten (devices)

   Um Anwendungen mit einem einfachen Zugriff auf die Hardware zu unterstützen, ermöglicht UNIX diese wie normale Datein zu verwenden. Es gibt zwei Arten von Einheiten in UNIX: (a) die block-orientierten Einheiten, die Daten in Blöcken (z. B. Festplatten) übertragen und (b) zeichenorientierten Einheiten die Datein auf einer Byte-für-Byte Basis (z.B. Modems und dumme (dumb) Terminale) übertragen.

4. Verknüpfungen (Links)

   Ein Link ist ein Zeiger auf eine Datei. Es gibt 2 verschiedene Links: (a) ein hartes Link daß nicht von der Datei unterscheidbar ist, (b) ein soft Link wird als eine Verzeichnisdatei die einen Pfad enthält implementiert.

Eine typische Unix Verzeichnisstruktur: eine hierarchische Baumstruktur



(Quelle: Wikipedia)

Unix Verzeichnisstruktur

`/`	Wurzel (root-) Verzeichnis, Systemsoftware
`/usr`	Softwarepakete des Herstellers
`/usr/local`	vom Admin installierte Software
`.../bin`	ausführbare Programmme
`.../lib`	Bibliotheken
`.../include`	Header-Dateien
`.../share`	Daten wie Icons, Sounds
`/home`	meist Lage der Benutzerverzeichnisse
`/media/...`	Temporär eingehängte Medien (CD, USB-Massenspeicher, ...)
`/dev`	Gerätedateien
`/proc`	(nur Linux) Systeminformationen

Beispiele: Pfade

- `~/.local/share/Trash`
  Ort des Mülleimers im eigenen Home-Directory, das `.local`-Verzeichnis ist versteckt
- `/home/mfyta/.local/Trash`
  Dasselbe Verzeichnis, wenn das Home-Directory `/home/mfyta` ist
- `.local/share/Trash`
  Auch dasselbe Verzeichnis, wenn ich im Home-Directory bin
- `share/Trash`
  . oder wenn ich im Verzeichnis „~/.local“ bin

Wildcards: Platzhälter für andere Zeichen

- Wildcards werden von der Shell aufgelöst (*Pattern-Matching*)
- Details hängen von der Shell ab (`man bash`!)
- „*“ passt auf jeden String, auch den leeren
- „?“ passt auf genau ein Zeichen
- „[a-zA-Z]“ passt genau auf die angegebenen Zeichen(bereiche)

Beispiel

Verzeichnis:

```
a abcd a.txt
b bc
.d d d3
```

Pattern	Treffer
*	`a abcd a.txt b bc d d3`
.	`a.txt`
*b?	`bc`
d	`abcd d d3`
[a-z][0-9]	`d3`
.*	`. .. .d`

Anführungszeichen (quotes)

Gewisse Sonderzeichen (z.B. '$\star$', '-', '{', usw.) sind in besonderer Weise vom Shell interpretiert. Um Argumente, die diese Zeichen verwenden, direkt (ohne Erweiterung des Dateinames) an Befehlen weiter zu geben, müssen Anführungszeichen genützt werden. Volgende 3 Zitierungsebenen können verwendet werden:

- Versuchen Sie ein '\' vor das Sonderzeichen einzufügen.
- Verwenden Sie doppelte Anführungszeichen (") um weitere Erweiterungen zu verhindern.
- Benutzen Sie einzelne Anführungszeichen (') für Argumente um weitere Erweiterungen zu verhindern.

Man kann auch das Anführungszeichen (‘) benutzen um die Ausgabe eines Befehles als Eingabe eines anderes Befehles zu geben, z.B.:

```
$ hostname
nestor
$ echo this machine is called ‘hostname‘
this machine is called nestor
```

Verknüpfungen/Links

Direkte (hart) und indirekte (symbolische oder Softlink) Verknüpfungen verweisen auf eine andere Datei oder ein anderes Verzeichnis.
In einem Unix-System kann man mit dem folgenden Befehl eine symbolische Verknüpfung erstellen:

- `ln -s /Quelldatei /Zieldatei(Optional)`

oder verständlicher

- `ln -s`

Es lässt sich auch ein Link namens „/home/wiki/nullink“ erstellen, der auf /dev/null zeigt:

- `ln -s /dev/null /home/wiki/nullink`

Ob die Datei „/home/wiki/nullink“ eine symbolische Verknüpfung ist, findet man mit einem dieser Befehle heraus:

- `file /home/wiki/nullink`
- `ls -l /home/wiki/nullink`

Dateien sortieren

→ `sort` Dateiname(n)
Der Befehl sortiert alphabetisch (oder numerisch mit der Option -n) die Zeilen in einer Datei. Die sortierte Ausgabe wird auf den Terminal aufgezeigt, und kann in einer anderen Datei gespeichert werden (durch Umleiten der Ausgabe).
`$ sort input1.txt input2.txt > output.txt`

- `uniq` Dateiname
Der Befehl entfernt die doppelt gegebene Zeilen aus einer Datei. Dieses ist besonders nützlich wenn mit `sort` kombiniert.
`$ sort input.txt | uniq > output.txt`

Datenkompression - Datensicherung (Backup) (I)

UNIX-Systeme unterstützen eine Reihe von Tools für Datensicherung und Datenkompression. Die nützlichsten sind:

- `tar` (tape archiver)
  Der Befehl sichert komplette Verzeichnisse und Dateien auf einem Bandlaufwerk oder (häufiger) in eine einzige Datenträger-Datei die als Archiv bekannt ist. Ein Archiv ist eine Datei, die andere Dateien sowie Informationen über diesen Dateien (Dateinamen, Eigner, Zeitstempel, Rechte) enthält. Mit diesem Befehl wird keine automatische Komprimierung durchgeführt.
  Um so eine Archiv-Datei zu erstellen:
  `$ tar -cvf Archivnamen`
  wobei die Archivnamen in der Regel eine `.tar` Erweiterung haben. Die `-c` Option bedeutet "erstellen", `v` verbose (die Dateinamen während der Archievirung auszugeben), und `f` bedeutet Datei (file). Um den Inhalt eines tar-Achivs zu sehen:
  `$ tar -tvf` Archivdateiname
  Um die Dateien aus einem tar-Archiv wiederherzustellen:
  `$ tar -xvf` Archivdateiname

Datenkompression - Datensicherung (Backup) (II)

- `cpio`: Mit diesem Befehl kann mann auch Archiven erstellen und lesen, aber der Inhalt von Verzeichnissen wird nicht automatisch archiviert. Es ist dann üblich `cpio` mit `find` zu kombinieren um ein Archiv zu erstellen:
  `$ find . -print -depth | cpio -ov -Htar > Archivdateiname`
  Dieser Befehl legt alle Dateien und Verzeichnisse im aktuellen Verzeichnis in eine Archivdatei ein. Die `-depth` Option steuert die Reihenfolge in der die Dateinamen erstellt werden und ist empfohlen um Probleme mit Berechtigungen während einer Zurückstellung zu verhindern. Die `-o` Option erstellt das Archiv, `-v` gibt die Dateienamen die archiviert werden und die `-H` Option gibt das Archivformat vor (in diesem Fall ein tar-Archiv).
  Der Befehl
  `$ cpio -tv <` Dateiname listet den Inhalt des Archivs auf.
  Um Dateien zurückzustellen, verwendet man:
  `$ cpio -idv <` Dateiname.
  Die `-d` Option erstellt Verzeichnisse nach Bedarf. Die `-u` Option zwingt die Zurückstellung von Dateien auf Dateien mit dem gleichen Namen (die den gleichen oder späteren

Datenkompression - Datensicherung (Backup) (III)

- `gzip`
  Dieses Dienstprogramm kann einzelne Dateien komprimieren und dekomprimieren (die nicht unbedingt Archivdateien seien müssen). Um Dateien zu komprimieren verwendet man:

  `$ gzip` Dateiname

  Die Datei wird dann gelöscht und durch eine komprimierte Datei die Dateiname.gz gennant wird erzetz. Umgekehrt, um die Datei wieder zurückzustellen:

  `$ gzip -d` Dateiname
  oder
  `$ gunzip` Dateiname.

Ein- und Ausgabe

- Ein Prozess hat wenigstens drei Dateien:

0	stdin	Eingabe (etwa Tastatur)
1	stdout	Standard-Ausgabe
2	stderr	Fehler-Ausgabe

- Wir können diese unabhängig umleiten:

`> <file>`	Umleiten von stdout in eine neue Datei
`>> <file>`	Wie >, aber hängt an
`>&`	Umleiten von stderr & stdout
`< <file>`	Liest Datei als stdin

- Reihenfolge ist wichtig!
- praktisch: Datei `/dev/zero` dient als leere Eingabe, `/dev/null`, um Ausgabe zu verschlucken

Umleiten von Ein- und Ausgabe

Die Ausgabe von Programmen ist in der Regel auf dem Bildschirm ausgegeben, während die Eingabe kommt in der Regel von der Tastatur (wenn keine Datei Argumente angegeben werden). Die Prozesse "schreiben" in der Regel auf die Standardausgabe (standard output) und bekommen die Eingabe von der Standardeingabe (standard input). Es gibt noch eine andere Ausgabe wo die Prozesse die so genannten Standardfehler (standard error) ihre Fehlermeldungen geben. Standardmäßig werden Fehlermeldungen ebenfalls auf dem Bildschirm angezeigt.
Um Standard-Ausgabe in eine Datei statt auf dem Bildschirm umzuleiten, verwenden wir den Operator >:

Beispiele: Umleiten von Ein- und Ausgabe (I)

- `grep Hase *.txt > hasen`
  Kopiert alle Zeilen, die „Hase“ enthalten, in Datei „hasen“
- `grep Igel *.txt >> hasen`
  Fügt alle Zeilen, die „Igel“ enthalten, an
- `ls *.txt >& errors > txt`
  Listet alle .txt-Dateien in Datei „txt“, Fehler in Datei „errors“
- `ls *.txt > txt >& errors`
  (!) Datei „txt“ ist leer, Fehler und .txt-Dateien in „errors“
- `grep Igel < hasen`
  Gibt alle Zeilen der Datei „hasen“ aus, die Igel enthalten
- `./myprogram < /dev/zero >& stderr > /dev/null &`
  Startet „myprogram“ im Hintergrund ohne Ein- oder Ausgabe, nur Fehler werden in „stderr“ gespeichert

Beispiele: Umleiten von Ein- und Ausgabe (II)

- ```
  $ echo hello
  hello
  $ echo hello > output
  $ cat output
  hello
  ```

  Der Inhalt der `output` Datei wird gelöscht, wenn die Datei bereits vorhanden ist. Wenn man die Ausgabe des `echo` Befehles in die Datei anhängen will, muss man den Operator » verwenden:

- ```
  $ echo bye » output
  $ cat output
  hello
  bye
  ```

  Um den Standardfehler zu erfassen, stellt man den Operator > mit eine 2 (in UNIX die Datei Zahlen 0, 1 und 2 werden an Standard-Input, Standard-Ausgabe und Standard-Fehler jeweils zugeordnet), zB:

- ```
  $ cat nonexistent 2>errors
  $ cat errors
  cat: nonexistent: No such file or directory
  ```
  ```

Umleiten von Ein- und Ausgabe (III)

Man kann noch den Standardfehler und die Standard-Ausgabe in zwei verschiedenen Dateien umleiten:

- `$ find . -print 1>errors 2>files`

oder in der gleiche Datei:

- `$ find . -print 1>output 2>output`

  oder

  `$ find . -print >& output`

Eine Standard-Eingabe kann auch mit dem Operator < umgeleitet werden, so dass die Eingaben aus einer Datei statt von der Tastatur gelesen wird:

- ```
  $ cat < output
  hello
  bye
  ```
  ```

Umleiten von Ein- und Ausgabe (IV)

Mann kann die Eingabeumleitung mit der Ausgabeumleitung kombinieren (nicht aber den gleichen Dateinamen verwenden), z.B.:

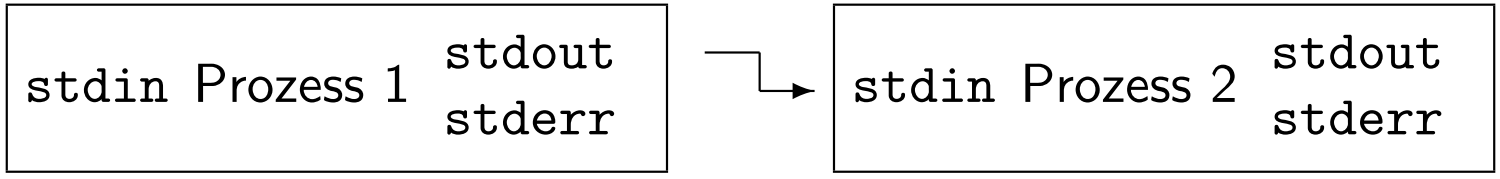
- `$ cat < output > output`

  Der Befehl löscht den Inhalt der ausgegebene Datei. Mit den Operator > , wird die Shell eine leere Datei bereit für die Ausgabe erstellen.
  Mann kann die Standardausgabe in System-Utilities weitergeben, die Dateinamen als "benötigen:

- `$ cat package.tar.gz | gzip -d | tar tvf -`

  Hier die Ausgabe des `gzip -d` Befehls wird als Eingabedatei für den `tar` Befehl benutzt.

Verknüpfen von Prozessen: Pipes



- „|“ verbindet die Ausgabe eines Prozesses mit der Eingabe eines anderen
- Mit „;“ können zwei Prozesse nacheinander gestartet werden

Beispiele

- `cd bla; ls`
  In Verzeichnis „bla“ wechseln und Dateien darin ausgeben
- `ps axww | grep bash | grep -v grep`
  Sucht alle laufenden bash-Shells, ohne den grep-Befehl auszugeben
- `ls -a | grep txt`
  Listet alle Dateien mit „txt“ im Namen

Pipes: Beispiele

Pipes sind für die Kombination von System-Utilities, um komplexere Funktionen auszuführen sehr nützlich. Zum Beispiel:

```
$ cat hello.txt | sort | uniq
```

Der Befehl erzeugt drei Prozesse (`cat`, `sort` und `uniq`), die gleichzeitig ausgeführt werden. Wenn diese ausgeführt werden, wird der Ausgang des `cat` Prozesses auf den `sort` Prozess weitergegeben, der dann dem `uniq` Befehl übergeben wird. `uniq` wird die Ausgabe auf dem Bildschirm (eine sortierte Liste der Benutzer in der die doppelt gegebenen Zeilen entfernt wurden) geben. Ebenso:

```
$ cat hello.txt | grep "hund grep-v "katze"
```

findet alle Zeilen in der hello.txt Datei, die die Zeichenkette 'hund' enthalten, aber nicht die Zeichenkette 'katze'.

Datei- und Verzeichnisrechte

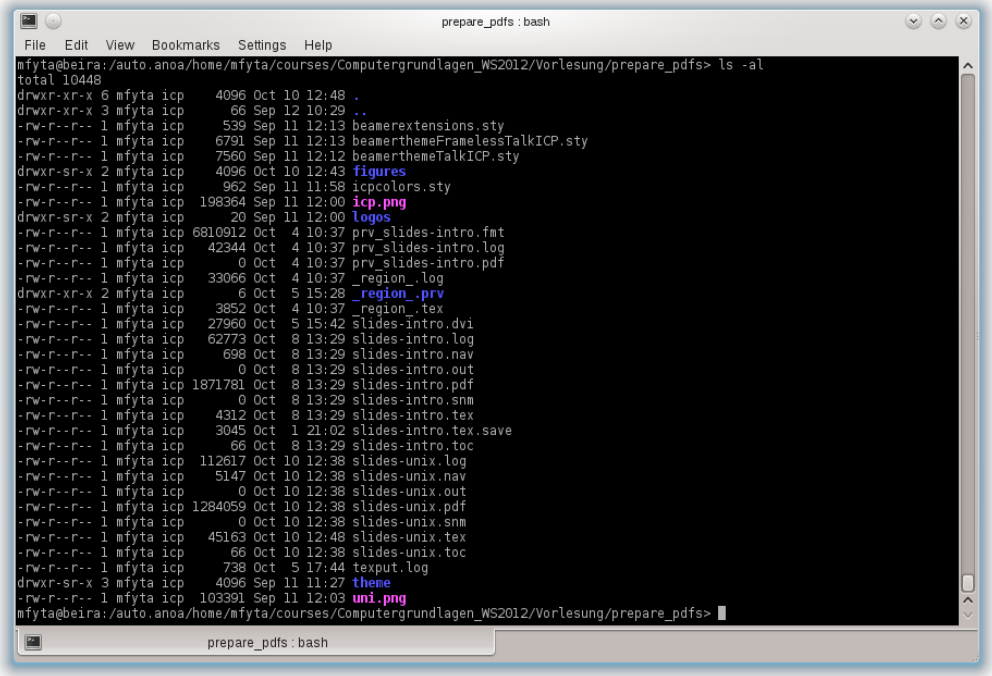
`ls -l <file>`	Rechte an einer Datei ausgeben
`chmod ugoa[=|+|-]rwx <file>`	Dateirechte ändern
`chgrp <group> <file>`	Datei-Gruppe ändern
`chown <user>:<group> <file>`	Dateibesitzer ändern

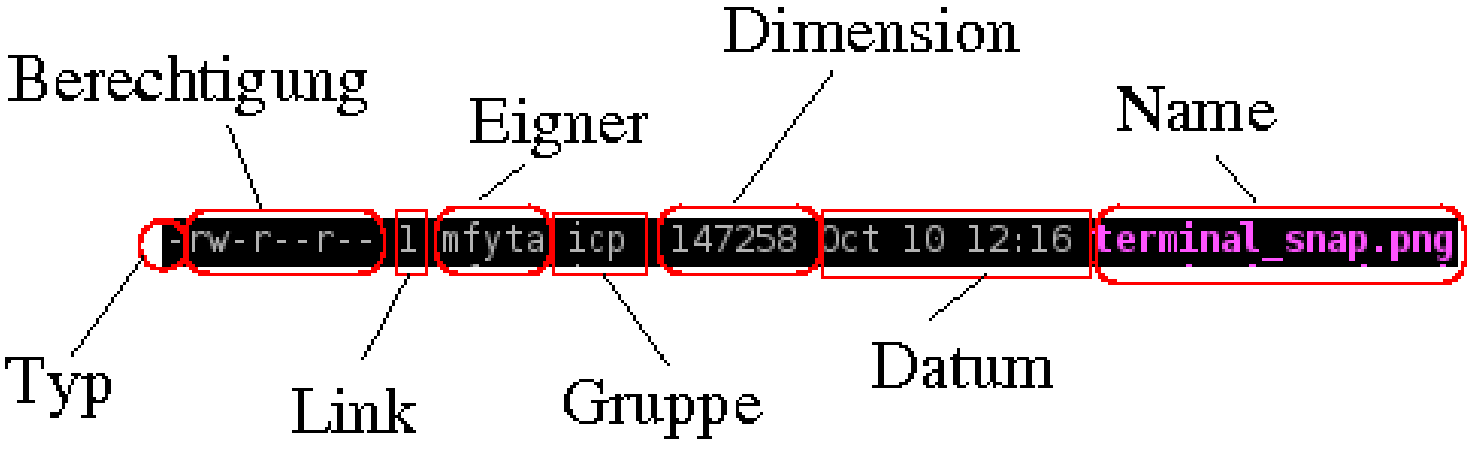
- Jede Datei gehört einem Benutzer und einer Gruppe (genauer, einer UID und Gruppen-ID)
- Rechte für Benutzer (`u`), Gruppe (`g`) und andere (`o`), oder alle (`a`)
- Kürzel für die meist benutzten Rechte:

Kürzel	bei Dateien	bei Verzeichnissen
`r`	lesen	Dateien anzeigen
`w`	schreiben	Dateien anlegen/löschen
`x`	ausführen	in das Verzeichnis wechseln

- Feinere Rechte durch ACLs (Zugriffskontrolllisten)

Dateiberechtigung (I)





Dateiberechtigung (II)

- Typ: 'd' (Verzeichnis), '-' (normale Datei), 'l' (symbolische Verknüpfung/Link), 'b'(block-orientiertes Gerät), 'c' (zeichen-orientiertes Gerät)
- 9 Berechtigungszeichen: 3 Zugrifftypen für 3 Benutzerkategorien: 'r' (read/lesen), 'w' (write/schreiben), 'x'(execute/ausführen)
- Benutzerkategorien: der Eigner (u), die Gruppe (g) und Sonstige (o).
- Link: Zahl der Dateisystemenverknüpfungen die auf eine Datei oder ein Verzeichnis hinweisen.
- Eigner: ist meistens der jeniger der die Datei oder das Verzeichnis erstellt hat.
- Gruppe: bezeichnet die Benutzer die an die Datei zugreifen können.
- Dimension: die Länge einer Datei oder die Anzahl der Bytes die das Betriebsystem benutzt um die Dateiliste in ein Verzeichnis zu speichern.
- Datum: das Datum an dem die Datei oder das Verzeichnis die zuletzt modifiziert wurden; ‘-u‘ zeigt wann die Datei oder das Verzeichnis zuletzt gelesen wurden.
- Name: der Name der Datei oder des Verzeichnisses.

Die Berechtigung ändern (I)

Die Berechtigung von Dateien und Verzeichnissen kann nur der Eigner oder der superuser (root) mit dem Befehl `chmod` (change mode) ändern:

```
$ chmod Optionen Dateien
```

Die Optionen kann man entweder als Oktalzahlen (jede Oktalziffer stellt die Berechtigung für den Benutzer, die Gruppe und Sonstige) oder symbolisch geben.

1. Oktalzahlen - Mapping der Berechtigung auf der entsprechend

---	0
--x	1
-w-	2
-wx	3
r--	4
r-x	5
rw-	6
rwx	7

z.B. der Befehl `$ chmod 600 test.txt`, setzt die folgende Bedingungen für `test.txt`: `rw- - - - - -`, das heisst dass nur der Eigner die Datei lesen und schreiben kann.

Die Berechtigung ändern (II)

2. Symbolische Optionen:
   Die folgende Zeichen werden benutzt:
   `u` (user/Benutzer),
   `g` (group/Gruppe),
   `o` (other/Sonstige),
   `a` (all/Alle),
   `r` (read/lesen),
   `w` (write/schreiben),
   `x` (execute/ablaufen),
   `+` (Rechte hinfügen),
   `-` (Rechte wegnehmen),
   `=` (Rechte zuteilen).

Die Berechtigung ändern (III)

Noch einige Beispiele

- Der Befehl `$ chmod ug=rw,o-rw,a-x *.txt`, setzt die Bedingungen `rw−rw− − −−` für alle Datein derren Name in *txt endet. Der Benutzer und die Gruppe können die Dateien lesen, in den Dateien schreiben. Sonstige haben gar keine Berechtigung auf diesen Dateien.

- Mit der Option `-R` can man rekursiv die Berechtigungen ändern. z.B. der Befehl `$ chmod -R go+r images` gewährt der Gruppe und Sonstigen Lese- und Schreibenrechte für alle Dateien und Verzeichnisse innerhalb von images.

- Der Befehl `$ chgrp`:
  `$ chgr` Gruppe Dateien
  ändert die Gruppe die eine Datei oder Verzeichniss besitzt. Der Befehl unterstützt auch die `-R` Option.

Beispiele: Dateirechte

- `mkdir test; chmod ug+rwx test`
  Legt ein Verzeichnis für die ganze Gruppe an
- `ls -ld test`
  `drwxrwxr-x 2 axel axel 4096 2010-10-24 18:53 test/`
- `chmod u-rwx test`
  Jetzt darf ich selber nicht mehr dran, aber die Gruppe!
- `echo "Hallo" >test/bla`

  `bash: test/bla: Permission denied`
- `chmod og-rwx ~/Documents`
  „Documents“ vor allen anderen (außer root) verstecken
- `chmod a+rx ~/bin/superprog`
  Das Programm „superprog“ für alle ausführbar machen

Benutzer

- Jeder Prozess und jede Datei gehört einem Benutzer
- Dieser wird meist von der ausführenden Shell festgelegt
- Jeder Benutzer gehört einer oder mehreren Gruppen an
- Ein Benutzer wird durch eine Zahl (UID) identifiziert
- nur auf *einem* Computer eindeutig (oder LDAP/NIS)
- Anmeldung meist mit Passwort

`w, who`	Wer ist gerade angemeldet?
`whoami`	Wer bin ich gerade?
`groups`	Meine Gruppen anzeigen
`passwd`	Passwort ändern
`su <user>`	Benutzer wechseln
`sudo <command>`	Befehl als root ausführen, wenn erlaubt

Benutzer: einige Befehle

- Abschaltung:`shutdown, halt, reboot` (in `/sbin`

  `/sbin/shutdown -r now` (jetzt abschalten und rebooten)
  `/sbin/shutdown -h +5` (in 5Min abschalten und halt)
  `/sbin/shutdown -k 17:00` (gefälschte Abschaltung um 17:00)

- `sync`: aktualisiert den Status des Dateisysteme

- Booten: `fsck` Dateisystem
  führt eine "Reparatur" des Dateisystems falls der Rechner nicht richtig abgeschaltet wurde.

- Benutzer hinfügen: `useradd` in `/usr/sbin`

  ```
  $ useradd tom
  $ passwd tom
  ```

Benutzergruppen kontrollieren

- `groupadd` (in `/usr/sbin`): erstellt eine neue Benutzergruppe und fügt die neue Information in `/etc/group` ein.

  `$ groupadd` Gruppenname

- `usermod` (in `/usr/sbin`): ändert die Gruppen Rechte eines Benutzers.

  `usermod -g` erste Gruppe Benutzername `-G` andere Gruppen

- `groups`: zeigt in welcher Gruppe ein Benutzer istgehört:

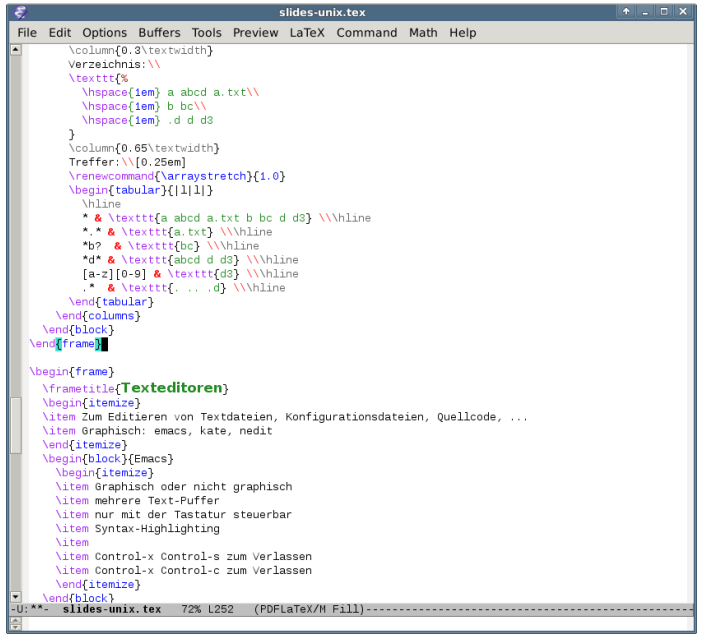
  `$ groups` Benutzername

Texteditoren

- Zum Editieren von Textdateien (Konfiguration, Quellcode, ...)
- unter UNIX wird fast alles über Textdateien gesteuert
- Graphisch: (x)emacs, nedit, kate, eclipse, ...

Emacs

- Graphisch oder nicht graphisch
- mehrere Text-Puffer
- komplett mit Tastatur steuerbar
- Spezialmodi für Python/C/LaTeX/...
- falls aus Versehen geöffnet: Control-x Control-c zum Verlassen
- Alternativ: xemacs, Aquamacs



Editor in der Shell: vi

- funktioniert in der Shell
- braucht außer Escape keine Sondertasten

  $\rightarrow$ tut auch, wenn emacs& Co. versagen (z. B. langsames Netz)!
- besser: vim (vi Improved), gvim (eigenes Fenster)

Befehlsmodus (Escape)

[N]dd	N Zeile(n) löschen
[N]yy	N Zeile(n) kopieren

p	Zeilen einfügen
h,j,k,l	Cursortastenersatz

Eingabemodus (i,a,o,...)

- Texteingabe, mit Escape beenden

Befehlsmodus (:)

x	speichern & beenden
s/A/B	A durch B ersetzen

q!	Beenden ohne Speichern
42	Zeile 42 anspringen