Computergrundlagen
Einführung in UNIX

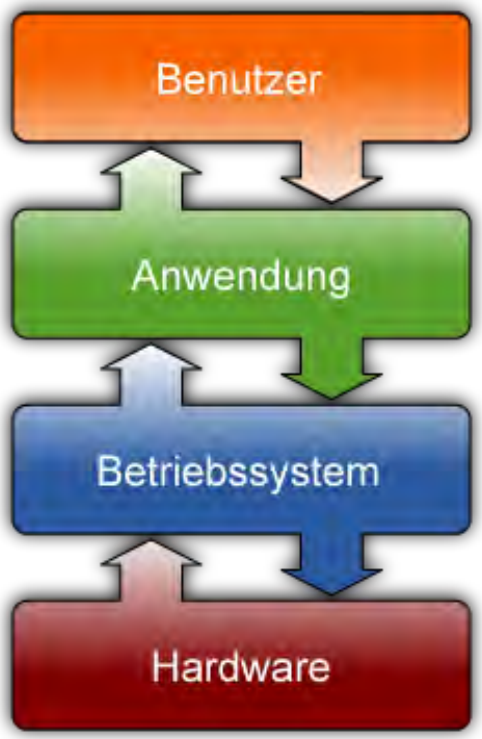
Institut für Computerphysik
Universität Stuttgart

Wintersemester 2018/19

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)

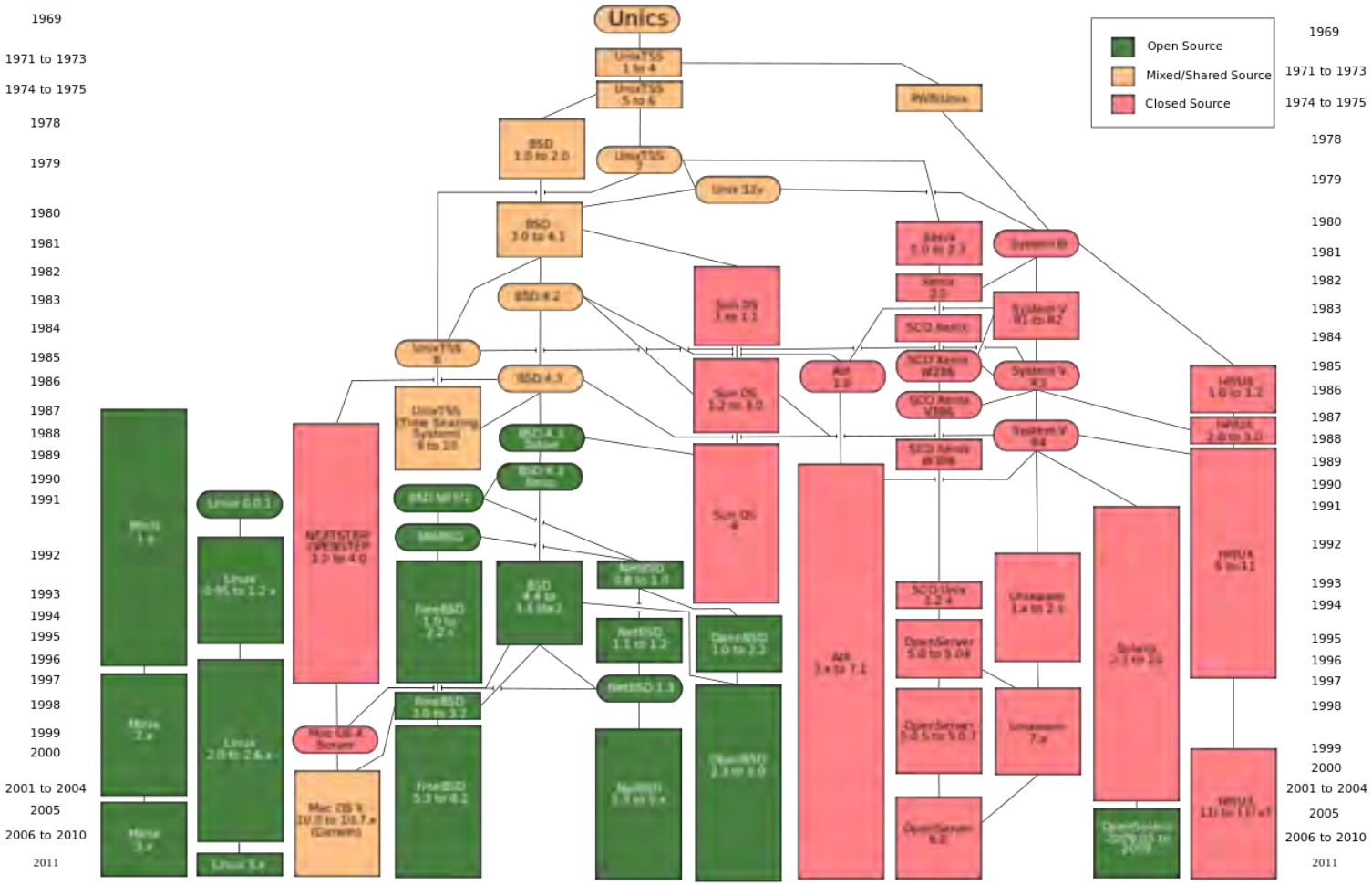
Betriebssysteme



- 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



(Quelle: Wikipedia)

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

Das Unix Dateisystem I

Umfasst Kernel, Programmdatei für alle Befehle, Information zur Konfigurierung, Benutzer Datei, spezielle Dateien zur Hardware und Betriebssystem Bedienung.

Vier Typen im Unix-artiges Dateisystem:

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

Das Unix Dateisystem II

1. Normale Dateien (ordinary files) enthalten:

   Text, Daten oder Programminformationen **aber** keine andere Dateien oder Verzeichnisse.

2. Ein Verzeichnis (directory) enthält:

   Dateien und andere Verzeichnisse.

- ! 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.
- ! Für Dateinamen lieber ’_’ statt ein Leerzeichen verwenden.

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
   Zwei Arten von Einheiten:

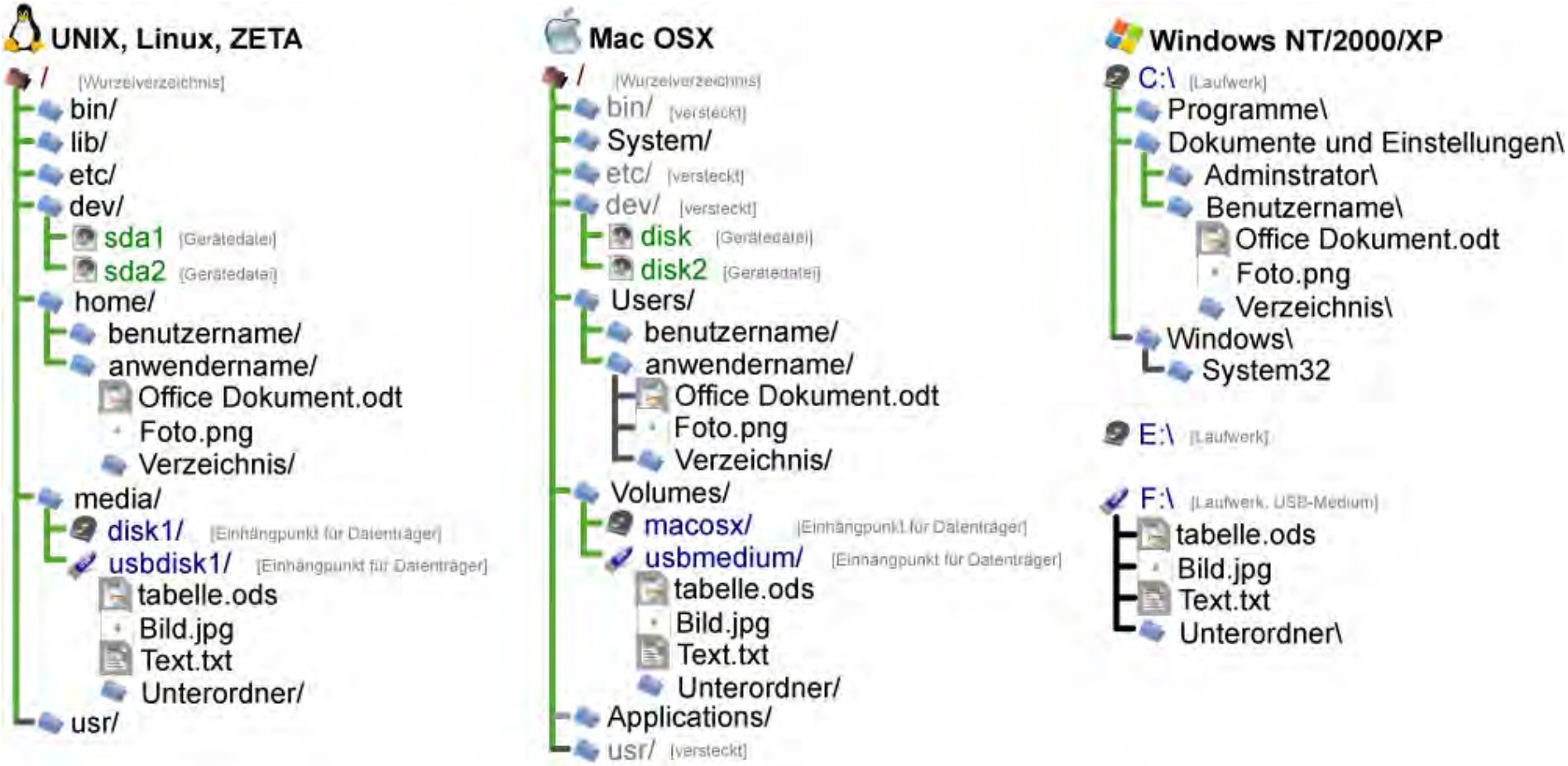
   (a) die block-orientierten Einheiten, die Daten in Blöcken (z. B. Festplatten) übertragen
   (b) zeichenorientierten Einheiten die Datein auf einer Byte-für-Byte Basis (z.B. Modems) übertragen.

4. Verknüpfungen (Links)

   Ein Link ist ein Zeiger auf eine Datei.
   Zwei Link Arten:

   (a) ein harter Link (von der Quell-Datei nicht unterscheidbar),
   (b) ein weicher (soft) Link (eine Verzeichnisdatei die einen Pfad enthält).

Eine typische Unix Verzeichnisstruktur (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

Architektur des Linux Betriebsystemes (I)

- Kernel
  Umfasst Gerätettreiber (Grafikkarte, Netzwerkkarte, Festplatten, usw.), Speicherverwaltung, Unterstützung für verschiedene Dateisysteme.
  Liegt in /boot/vmlinuz (binäre Form) und in /usr/src/linux (Quelldatei)
- Shells und GUIs
  Linux unterstützt 2 Befehleingaben:
  1. zeichenorientierte Befehlszeilen (Kommandozeilen/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.

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),
    - usw.

Unix Systeme: Ein- und Ausloggen (I)

1. Zeichenorientierte Terminale (TTY):

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

   - Benutzername eingeben und 'enter/return' (↩) (Unix ist schreibungsabhängig). Dann benötigt das System das Kennwort:
     `login:mfyta`
     `password:`
   - Man tippt das Kennwort, dann wieder die Taste enter/return ↩. 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

   Grafisches Prompt für Benutzername und Kennwort. Ein grafischer 'window manager' wird hervorgerufen (ähnlich wie bei MSWindows) mit Menüs oder Iconen für "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.)

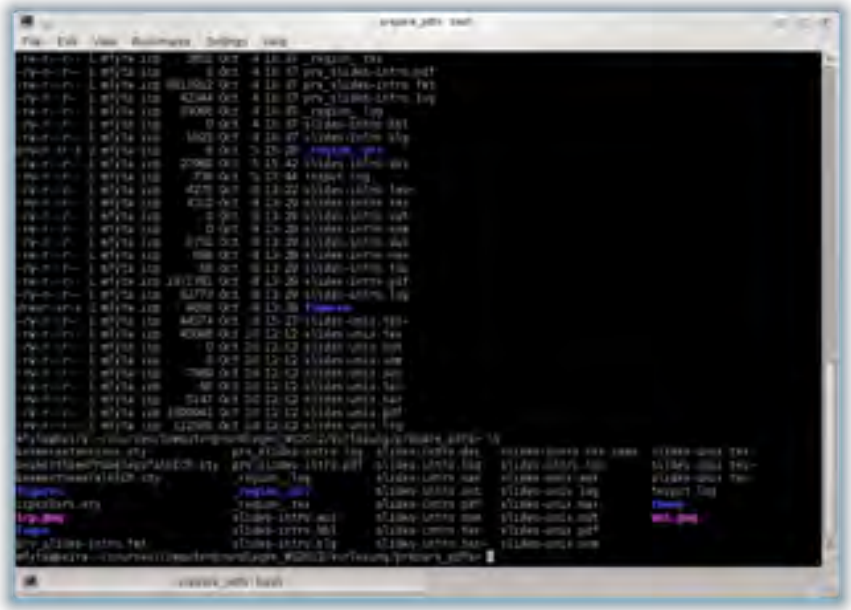
Allgemeines Format der Unix Befehle

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 ↩
```

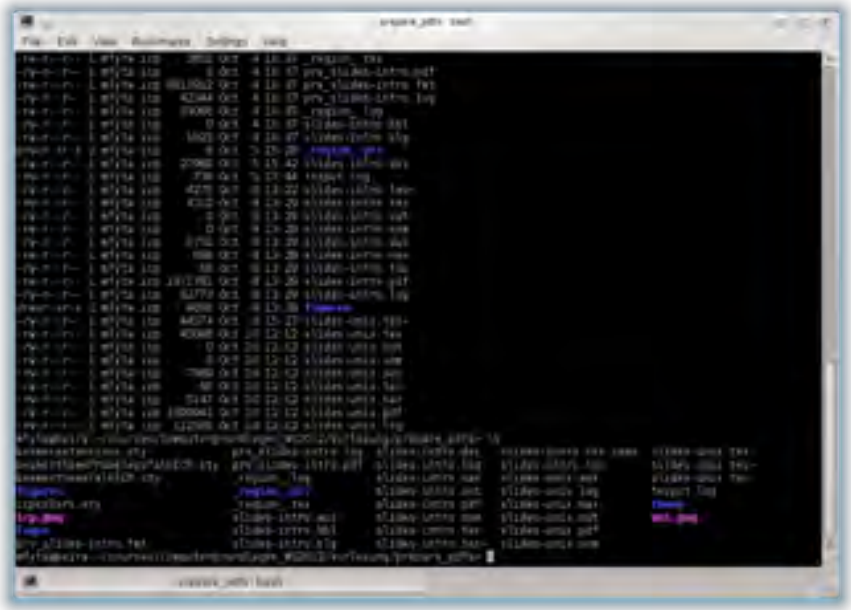
Hier “command”(Befehl) darf ein Verb, “options” (Optionen) ein Adverb und “targets”(Ziele) die direkt Objekte eines Verbes sein. Mehrere Optionen müssen können alle nach einem Halbgeviertstrich (’-’) aufgelistet werden.

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 Programm) hat ein aktuelles Arbeitsverzeichnis
- Pfade, die nicht mit „ / “ oder „ ~ “ anfangen, sind *relativ* zum Arbeitsverzeichnis

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
  → 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

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.) werden in einer besonderen Art und Weise von der 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:

- Einen '\' vor dem Sonderzeichen einfügen.
- Doppelte Anführungszeichen (") verwenden um weitere Erweiterungen zu verhindern.
- Einzelne Anführungszeichen (') für Argumente verwenden um weitere Erweiterungen zu verhindern.

Dateien finden (I)

1. `find`: [man hat ungefähr eine Vorstellung der Verzeichnisstruktur]

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

   Der Befehl sucht die Zieldatei überall unter dem gegebenen Verzeichnis.

2. `which`: [um herauszufinden genau welches Anwendungsprogramm (und wo es gespeichert ist) mit einen bestimmten Namen aufgerufen ist]

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

3. `locate`: [sehr schneller als `find` wenn man Dateien sucht deren Namen einen bestimmten Zuchbegriff hat oder ein "grosser"Dateiraum durchgesucht (z.B. unter \) wird]

   `$ locate ‘‘.txt’’`

   Der Befehl findet alle Dateinamen im Dateisystem die in ".txt” enden.

Dateien finden (II)

→ `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. Allerding findet er Dateien die vor kurzem gelöscht wurden.

Text in Dateien finden - `grep`

→ `grep` (General Regular Expression Print)

- `$ grep` Optionen Suchbegriff Datei(en)g

  Der Befehl sucht die angegebenen Dateien (oder die Standard Eingabe, wenn keine Dateien bennant werden) für Zeilen die zu einem bestimmten Suchbegriff passen.

  z.B. `$ grep hello *.txt`

  durchsucht alle Text-Dateien im aktuellen Verzeichnis für "helloSZeilen.

  Einige nutzbare Optionen:

  `-c`: zält die Anzahl der entsprechenden Zeilen auf,
  `-i`: Groß-/Kleinschreibung wird ingoriert,
  `-v` die Zeile die nicht passen werden ausgegeben,
  `-n` Ausdruck der Zeilenummer vor der entsprechenden 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) enthalten.

Falls man Dateien in einem gesamten Verzeichnisbaum für 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 "helloSZeilen.

Suchen und Finden: Zusammenfassung

grep: IN Dateien suchen

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

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

find: Dateien suchen

```
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 zugegriffen/geändert wurden

Datenkompression - Datensicherung (Backup) (I)

UNIX-Systeme unterstützen eine Reihe von Tools für Datensicherung und Datenkompression. Beispiele:

- `tar` (tape archiver)
  Der Befehl sichert komplette Verzeichnisse und Dateien auf einem Bandlaufwerk oder (häufiger) in einer einzigen Datenträger-Datei die als Archiv dient. Ein Archiv ist eine Datei, die andere Dateien sowie Informationen über diesen Dateien (Dateiname, Benutzer, 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 ërstellen", `v` verbose (soll die Dateinamen während der Archievirung ausgeben), 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)

- `gzip`

  Dieses Dienstprogramm kann einzelne Dateien komprimieren und dekomprimieren (müssen nicht unbedingt Archivdateien sein).

  Um Dateien zu komprimieren verwendet man:

  `$ gzip` Dateiname

  Die Datei wird dann gelöscht und durch eine komprimierte Datei (die Dateiname.gz) erzetz.

  Umgekehrt, um die Datei wieder zurückzustellen:

  `$ gzip -d` Dateiname

  oder

  `$ gunzip` Dateiname.

Datei- und Verzeichnisrechte

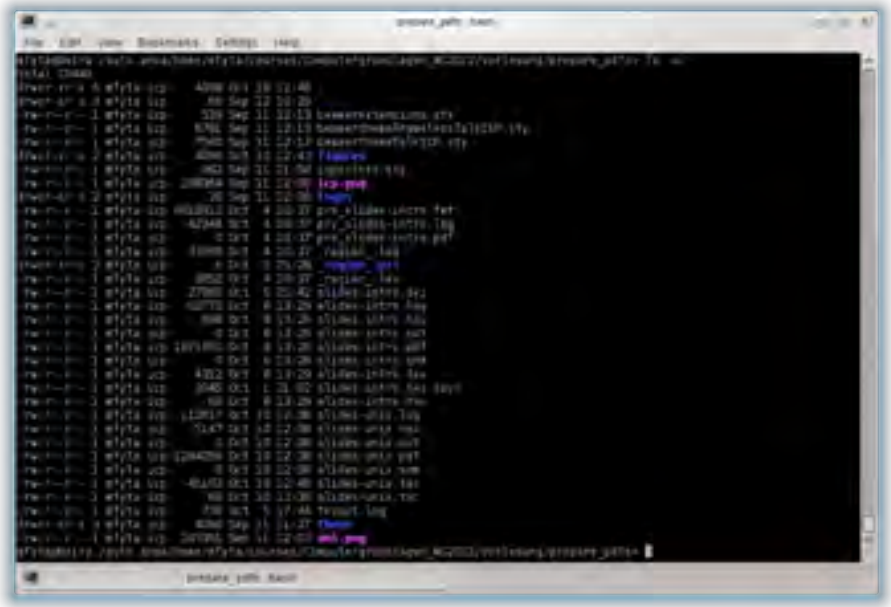
`ls -l <file>`	Rechte an einer Datei ausgeben
`chmod ugoa[=|+|-]rwx <file>`	Dateirechte ändern
`chgrp <group> <file>`	Datei-Gruppe ändern
`chown <user>:<group> <file>`	Dateibesitzer ändern

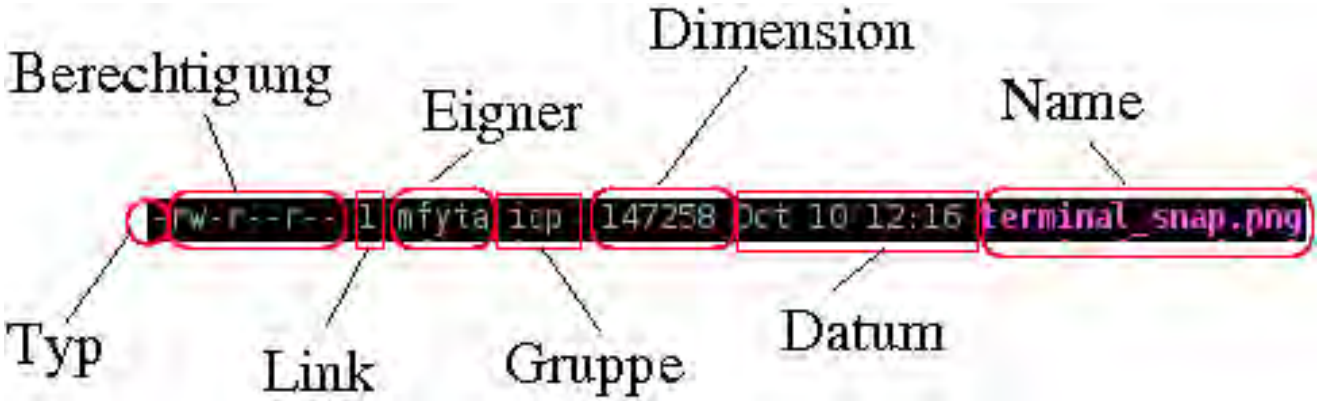
- Jede Datei gehört einem Benutzer und einer Gruppe (genauer, einer UID und Gruppen-ID)
- Rechte für Benutzer (`u`), Gruppe (`g`) und andere (`o`), oder alle (`a`)
- Kürzel für die meist benutzten Rechte:

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)

Datei- und Verzeichnisrechte (II)





Datei- und Verzeichnisrechte (III)

- 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.

Änderung der Datei- und Verzeichnisrechte (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 entsprechende Oktalziffer:

---	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- --- --- ---` [nur der Eigner darf die Datei lesen und schreiben].

Änderung der Datei- und Verzeichnisrechte (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).

Änderung der Datei- und Verzeichnisrechte (III)

Weitere Beispiele:

- Der Befehl $ `chmod ug=rw,o-rw,a-x *.txt`, setzt die Bedingungen `rw-rw- --- --` für alle Dateien mit dem Namen "*txt. Der Benutzer und die Gruppe dürfen lesen und schreiben. Sonstige Benutzer 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 Schreibrechte für alle Dateien und Verzeichnisse innerhalb von ïmages".
- Der Befehl `$ chgrp`:

  `$ chgr` Gruppe Dateien

  ändert die Gruppe einer Datei, bzw. eines Verzeichnisses. 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 user user 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

Aliases: Abkürzungen

`alias <short>='<cmd>' ...`	Alias setzen
`unalias <short>`	Alias löschen
`alias`	Alle definierten Aliases ausgeben

- Aliases erlauben verkürzte Befehlseingabe
- auch, um Befehlen Default-Parameter zu geben

Beispiele

- `alias cp='cp -i'  mv='mv -i'  rm='rm -i'`
  Löschen und überschreiben nur auf Nachfrage (!)
- `alias ls='ls --color=auto'`
  Dateien bei Ausgabe nach Typ einfärben
- `alias ll='ls -l'`
  Ausführliche Ausgabe

Das `sed` Unix-Werkzeug

- `sed` steht für Stream Editor und ist ein Unix-Werkzeug, mit dem Texte geändert werden können.
- Anwendung: `sed 's/alt/neu/g' Eingabedatei >Ausgabedatei`
- Beispiele:
  1. `$ sed 's/foo/bar/g' foo.txt > bar.txt` : ersetzt 'foo' mit 'bar' in der foo.txt Datei
  2. `$ sed '/baz/s/foo/bar/g' foo.txt` : ersetzt 'foo' mit 'bar' in der foo.txt Datei nur in den Zeilen die 'baz' enthalten.
  3. `$ sed '/baz/!s/foo/bar/g'` : ersetzt 'foo' mit 'bar' in der foo.txt Datei NICHT in den Zeilen die 'baz' enthalten.
  4. `$ sed 's/blau/rot/g;s/gruen/rot/g;s/gelb/rot/g' bunt.txt`: erzetzt 'blau' oder 'gruen' oder 'gelb' mit 'rot'
  5. `$ sed '/sonne/d' in.txt > out.txt` : löscht alle Zeilen die 'sonne' enthalten.
  6. `$ generate_data | sed -e 's/x/y/g'`
  7. `$ cat zoo.dat |awk '{print$2}' | sed 's/loewe/tiger/g'` : gibt die 2. Spalte der Datei 'zoo.dat' aus und ergänzt die Zeichenkette 'loewe' mit 'tiger'.

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
```

Verknüpfungen/Links

Direkte (harte) oder indirekte (symbolische oder Softlink) Verknüpfungen verweisen auf eine andere Datei, bzw. ein anderes Verzeichnis.
In einem Unix-System kann man wie folgend 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`

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 gegeben.
- 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**). Fehlermeldungen (**standard error**) werden auf dem Bildschirm angezeigt.
- Standard-Ausgabe in einer Datei statt auf dem Bildschirm umzuleiten ist mit dem Operator ’>’ möglich.

Beispiele: Umleiten von Ein- und Ausgabe (II)

- `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 (III)

- ```
  $ 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` Befehls 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 (IV)

Umleitund des Standardfehlers und der Standard-Ausgabe in zwei verschiedenen Dateien:

- `$ 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 Eingabe aus einer Datei (nicht der Tastatur) gelesen wird:

- `$ cat < output`

  `hello`

  `bye`

Umleiten von Ein- und Ausgabe (V)

Kombination der Eingabeumleitung und Ausgabeumleitung (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.

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

  Die Standardausgabe in system utilities weitergeben; die Dateinamen als "-" weitergeben.
  Hier die Ausgabe des `gzip -d` Befehls wird als Eingabedatei für den `tar` Befehl benutzt.

Prozesse

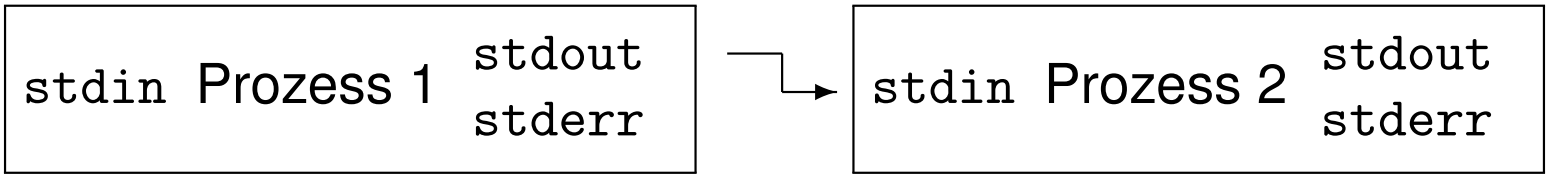
- Ein Prozess ist ein Programm das ausgeführt wird.
- Alle UNIX-Prozesse werden durch eine Prozess-ID oder PID identifiziert.
- Jedes Mal, wenn aus der Shell ein Anwendungsprogram aufgerufen wird, werden ein oder mehrere childProzesse von der Shell erstellt.
- Ein wichtiger Prozess ist der init-Prozess; der erste Prozess (in der Regel hat PID 1) der ausgefürt wird wenn ein UNIX-System startet. Alle anderen Prozesse folgen diesen.

Prozesse (II)

- Multitasking - (fast) beliebig viele Prozesse „gleichzeitig“
- Jedes Programm ist ein eigener Prozess
- Control-z stoppt den aktuell laufenden Befehl
- ein Befehl, der mit „&“ endet, wird im Hintergrund gestartet
- ein Hintergrundprozess erhält keinen Tastaturinput, aber schreibt auf das Terminal
- Grundlegende Prozessbefehle:

`ps`	Anzeige der aktuell laufenden Prozesse
`top`	fortlaufende Anzeige der aktivsten Prozesse
`bg`	gestoppten Prozess in den Hintergrund
`fg`	... und wieder in den Vordergrund
`kill <pid>`	Prozess beenden

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.

    Die Ausgabe des `cat` Prozesses wird dem `sort` Prozess weitergegeben, der dann dem `uniq` Befehl übergeben wird.

    `uniq` wird die Ausgabe (eine sortierte Liste der Benutzer in der die doppelt gegebenen Zeilen entfernt wurden) auf dem Bildschirm zeigen.
  - `$ 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'.

Prozesse mit der aktuellen Shell verbunden (I)

- Die meisten Shells bieten anspruchsvolle Job-Kontrolle, mit der man viele ausgeführte Aufträge (dh Prozesse) zur gleichen Zeit kontrolieren kann. Dies ist nützlich, wenn man zum Beispiel eine Textdatei bearbeitet und diese Bearbeitung aussetzen will um etwas anderes zu tun. Mit der Job-Kontrolle, kann man den Editor aussetzen, und zurück zur Shell-prompt gehen und an etwas anderes zu arbeiten. Wenn man fertig ist, kann man wieder mit den Editor weitermachen, als ob keine Aussetzung passiert ist.

- Jobs können entweder im Vordergrund oder im Hintergrund ausgeführt werden. Es kann nur einen Job in den Vordergrund zu jeder Zeit geben. Der Vordergrund Job hat die Kontrolle über die aktuelle Shell. Dieser bekommt eine Tastatur-Eingabe und gibt die Ausgabe auf dem Bildschirm. Jobs im Hintergrund erhalten keine Eingabe vom Terminal, und werden in der Regel ruhig ausgefürt.

Prozesse mit der aktuellen Shell verbunden (II)

- Der Vordergrund Job kann mit `Ctrl-Z` ausgesetzt werden, dh vorübergehend gestoppt werden. Ein ausgesetzter Job kann im Vordergrund oder im Hintergrund weiterlaufen durch Eingabe von "`fg`" oder "`bg`". Die Aussetzung eines Jobs ist nicht das gleiche wie die Unterbrechung (mit `Ctrl-C`). Unterbrochene Jobs werden permanent unterbrochen und können nicht wieder fortgesetzt werden.

- Hintergrund-Jobs können auch direkt von der Kommandozeile ausgeführt werden, durch ein '`&`'-Zeichen in der Befehlzeile. z.B.:

```
$ find / -print 1>output 2>errors &
[1] 27501
```

  Die `[1]` ist die Auftragsnummer des Prozesses im Hintergrund und die Zahl `27501` ist die PID des Prozesses.

Prozesse mit der aktuellen Shell verbunden (III)

- Erstellung einer Liste aller Jobs die mit der aktuellen Shell verbunden sind:

  ```
  $ jobs [1]+ Running find / -print 1>output 2>errors &
  ```

- Wenn man mehr als einen Job hat, kann man dem Job `%n`, wobei n die Job-Nummer, zuweisen.
  - `$ fg %3`

  bringt den Job mit Job-Nummer 3 im Vordergrund.
- Um die Prozess-ID der laufenden Prozessen die mit der Shell und der Jobs verbunden sind herauszufinden, verwenden man `ps`:

```
$ ps
 PID TTY TIME CMD
17717 pts/10 00:00:00 bash
27501 pts/10 00:00:01 find
27502 pts/10 00:00:00 ps
```

Hier die PID der Shell (bash) ist 17717, die PID für `find` ist 27501 und die PID für `ps` ist 27502.

Prozesse mit der aktuellen Shell verbunden (IV)

- Einen Prozess oder einen Job mit `kill` beenden, entweder mittels der PID oder Job-Nummer:

```
$ kill %1
```

oder

```
$ kill 27501
```

- Der Befehl `kill` sendet dem Prozess nur ein Signal zur Anforderung einer Beendigung (funktioniert nicht immer).
- Um einen Prozess mit eine höhere Wahrscheinlichkeit (erzwingend) zu beenden, gibt es die `-9` Option (das SIGKILL signal):

```
$ kill -9 27501
```

- Der Befehl `kill` kann verwendet werden um auch andere Arten von Signalen den laufenden Prozessen su übersenden. z.B. die Option `-19` (SIGSTOP) unterbricht einen laufenden Prozess. Um eine Liste von solchen Signalen zu sehen: `kill -l`.

Prozesse - Kontrolle

- `ps` zeigt nicht nur die Prozesse in der aktuellen Shell sondern auch alle Prozesse die auf dem Rechner laufen:

  `$ ps -fae` oder `ps -aux`

  `ps -aeH` zeigt die volständige Prozess-Hierarchie.
- `top`: interaktive Beobachtung der Aktivität des Systems.

  Detaillierte Statistik über die aktuell laufenden Prozesse werden angezeigt und ständig aktualisiert.

  Prozesse werden in der Reihenfolge der CPU-Verwendung angezeigt.

  Nützliche Tasten sind:

  `s` - die Frequenz für die Aktualisierung

  `k` - den Prozess (durch der PID) beenden

  `u` - die Prozessen eines Benutzers anzeigen

  `q` - beenden
- `pkill`: beendet Prozesse nach den Namen (nicht PID/Job-Nummer).

  ```
  $ pkill find
   [1] + Terminated find /-print 1> output 2> errors
  ```

  Beendet den Hintergrund `find` Prozess (zusammen mit allen anderen laufenden `find` Prozessen).
- Aus Sicherheitsgründen darf man nur eigene Prozesse beenden (sofern man nicht admin ist).

Rückgabewerte

- Wie kann ich auf einen Fehler reagieren?
- Prozesse signalisieren Probleme durch Rückgabewert
- 0: Erfolg, >0: kein Erfolg
- Wert kann durch „$?“ in der Shell abgefragt werden
- `proc1 && proc2`
  proc2 startet genau dann, wenn proc1 keinen Fehler meldet
- `proc1 || proc2`
  proc2 startet genau dann, wenn proc1 einen Fehler meldet

Beispiele

- `ls *.txt || echo "Keine .txt-Datei hier"`
- `test -f "bla" && cat bla`
  Gibt die Datei „bla“ nur aus, wenn sie existiert

Shells

In der Informatik bezeichnet man als Shell die Software, die den Benutzer in irgendeiner Form mit dem Computer verbindet

Unix Shells

- Bourne Shell (sh)
  - Bourne-Again shell (bash)
  - Korn shell (ksh)
- C shell (csh)
  - TENEX C shell (tsch)
- Emacs shell (eshell)
- etc...

Beispiel: Bash (oft die Standard-Shell)

Bash ist ein Kommandozeileninterpreter, bzw. ein Computerprogramm, das eine Zeile Text von einer Kommandozeile (command-line interface) einliest und als Kommando (Anweisung, Befehl) interpretiert. Das Kommando wird dann ausgeführt und das Ergebnis dem Benutzer angezeigt. Bash kann auch Befehle aus eine Datei (Skript) lesen.

Shells: relevante Dateien

	sh	ksh	csh	tcsh	bash
/etc/.login			login	login	
/etc/csh.cshrc			yes	yes	
/etc/csh.login			login	login	
~/.tcshrc				yes	
~/.cshrc			yes	yes[4]	
$ENV (typically ~/.kshrc) [5]		yes			
~/.login			login	login	
~/.logout			login	login	
/etc/profile	login	login			login
~/.profile	login	login			login[6]
~/.bash_profile					login[6]
~/.bash_login					login[6]
~/.bash_logout					login
~/.bashrc					int.+n/login

(Quelle: Wikipedia)

Skripte

- Wie kann ich mir komplexe Befehle merken?
- Gar nicht – aber der Computer kann es für mich!
- Einfach die Befehle in eine Textdatei schreiben und ausführbar machen
- „#!“ (Shebang) in der ersten Zeile bestimmt ausführende Shell
- Kommandozeilenwerte als „$1“, „$2“, ...

Beispiel

- Datei `mvtonewdir.sh` erstellen mit Inhalt:

```
#!/bin/bash
mkdir -p $2 && chmod og-rwx $2 && mv $1 $2
```

- `chmod a+rx mvtonewdir.sh`
  ausführbar machen
- `./mvtonewdir.sh bla test`
  und ausprobieren

Shell-Variablen

- In der Shell kann man Werte mit „=“ speichern und mit „ $ “ auslesen

```
name=bla; echo "name ist $name"
```

```
name ist bla
```

- Leerzeichen trennen Befehle, müssen daher in Hochkommata

  Falsch: `i=Meine Photos`

```
Photos: command not found
```

  $\Longrightarrow$ setzt `i=Meine` und führt dann Befehl `Photos` aus

  Richtig: `i="Meine Photos"`

- es darf kein Leerzeichen vor oder nach „=“

  Falsch: `i = 1`

```
i: command not found
```

  $\Longrightarrow$ führt Befehl `i` mit Parametern `=` und `1` aus

  Richtig: `i=1`

Shell-Variablen

- Skript-Parameter verfügbar als $1,$2,...

```
eingabe="$1"; echo $eingabe
```

- $0 ist Skriptname
- $* ist Liste alle Parameter

```
echo "usage: $0 <file> <dir>, not $*"
```

```
usage: mvtonewdir.sh <file> <dir>, not a b c
```

- $? gibt den Rückgabewert des letzten Befehls
- Ausgabe eines Befehls per „ ‘ “ (Backtick) als String

```
files=‘find . -name "*.txt"‘; echo $files
```

```
./text1.txt ./text2.txt
```

$\Longrightarrow$ Variable `files` enthält die Standardausgabe des `find`-Befehls

Umgebungsvariablen

`export <var>[=<value>]`	Variable exportieren
`unset <var>`	Variable löschen

- Exportierte Variablen (Umgebungsvariablen) stehen allen aufgerufene Programmen zur Verfügung
- Einige oft benutze Umgebungsvariablen:

`PWD`	aktuelles Verzeichnis
`HOME`	Pfad zum Benutzerverzeichnis
`DISPLAY`	X-Server (meist lokal, „ :0.0 “)
`PS1`	Prompt (z.B. „\u:\W “, Benutzer + Verzeichnis)
`EDITOR`	Standard-Texteditor (meist „ vim “)
`LANG`	Spracheinstellung („ de_DE “ oder „ en_US.UTF-8 “)

Wichtige Umgebungsvariablen

`PATH`: Befehlspfad

- Liste der Verzeichnisse, die Programme enthalten
- Trennzeichen ist „ : “
- Sollte aus Sicherheitsgründen *niemals* . enthalten
- Vom System vorbereitet, vom Benutzer ergänzt
- Beispiel:

```
PATH="~/bin:$PATH"; echo $PATH
/home/user/bin:/usr/local/bin:/usr/bin:/bin:/usr/games
```

`LD_LIBRARY_PATH`: Bibliothekspfad

- Dasselbe für dynamische (shared) Bibliotheken (`.so`-Dateien)
- Oft bei selbstcompilierter Software nötig

Musterersetzung

`${<var>%<pat>}`	`pat` am Ende entfernen
`${<var>%%<pat>}`	wie oben, aber längster Treffer
`${<var>#<pat>}`	`pat` am Anfang entfernen
`${<var>##<pat>}`	wie oben, aber längster Treffer
`${<var>//<abc>/<def>}`	`abc` durch `def` ersetzen

Beispiele (`TEST=/home/user/abc.txt.old`)

`${TEST%.*}` → `/home/user/abc.txt`

`${TEST%%.*}` → `/home/user/abc`

`${TEST##*/}` → `abc.txt.old`

`${TEST//ab/de}` → `/home/user/dec.txt.old`

Einfache Schleifen

```
for <var> in <list>; do <cmd>; done
```

- Für jeden Wert in der Liste wird die Variable auf den Wert gesetzt und das Kommando ausgeführt
- Leerzeichen ist Trennzeichen in der Liste

Beispiele

- ```
  for f in 1 2 3 4; do echo -n "$f,"; done; echo
  ```
  ```
  1,2,3,4,
  ```
- ```
  for f in *.txt; do
      n=${f%.txt}-2.txt; echo "$f -> $n"; mv $f $n
  done
  ```
  ```
  text1.txt -> text1-2.txt
  text2.txt -> text2-2.txt
  ```
  $\Longrightarrow$ Benennt alle Textdateien um und gibt aus, was es tut

Bedingte Anweisungen

```
if [!] <cond>; then <cmd1>; [ else <cmd2>; ] fi
```

- Führt `cmd1` genau dann aus, wenn der Befehl `<cond>` 0 zurückgibt
- `cmd2` genau dann, wenn der Befehl `<cond>` nicht 0 zurückgibt
- `!` dreht die Bedingung genau um

Beispiele

- `if test -f $f; then echo "$f gibt es"; fi`
  Test, ob `$f` eine Datei ist
- `if ! test -f $f; then echo "$f fehlt"; fi`
  Gibt nur etwas aus, wenn es `$f` nicht gibt
- ```
  if test -f $f; then
    if grep -q bla $f; then echo "in $f ist bla"; fi
  else echo "$f fehlt"; fi
  ```
  Meldet, ob `$f` „bla“ enthält oder nicht existiert
  ```

Beispiel: Ein einfaches Shell-Programm

Erzeugen einer Liste von Bildern für eine Desktop-Slideshow aus allen Verzeichnissen, die eine versteckte Datei „ .background “ enthalten:

```
#!/bin/sh
BACK=~/.config/xfce4/desktop/backdrop.list
echo "# xfce backdrop list" >$BACK

for dir in /home/user/Pictures/*; do
  if test -d "$dir" -a -f "$dir/.background"; then
    find "$dir" -mindepth 1 -maxdepth 1 »$BACK
  fi
done
```

Tipp: `touch <file>` erzeugt bequem eine leere Datei

Die awk Skriptsprache

```
awk Bedingung { Anweisungsblock }
```

- Eine Programmiersprache zur Bearbeitung und Auswertung strukturierter Textdaten
- Die Sprache arbeitet fast ausschließlich mit dem Datentyp Zeichenkette (String). Daneben sind assoziative Arrays (d. h. mit Zeichenketten indizierte Arrays) und reguläre Ausdrücke grundlegende Bestandteile der Sprache.
- Der Benutzer kann Variablen innerhalb von Anweisungsblöcken durch Referenzierung definieren, eine explizite Deklaration ist nicht notwendig.
- Befehl: Die Syntax der Befehlsanweisungen von awk ähnelt derjenigen der Programmiersprache C.

awk - Beispiele (print)

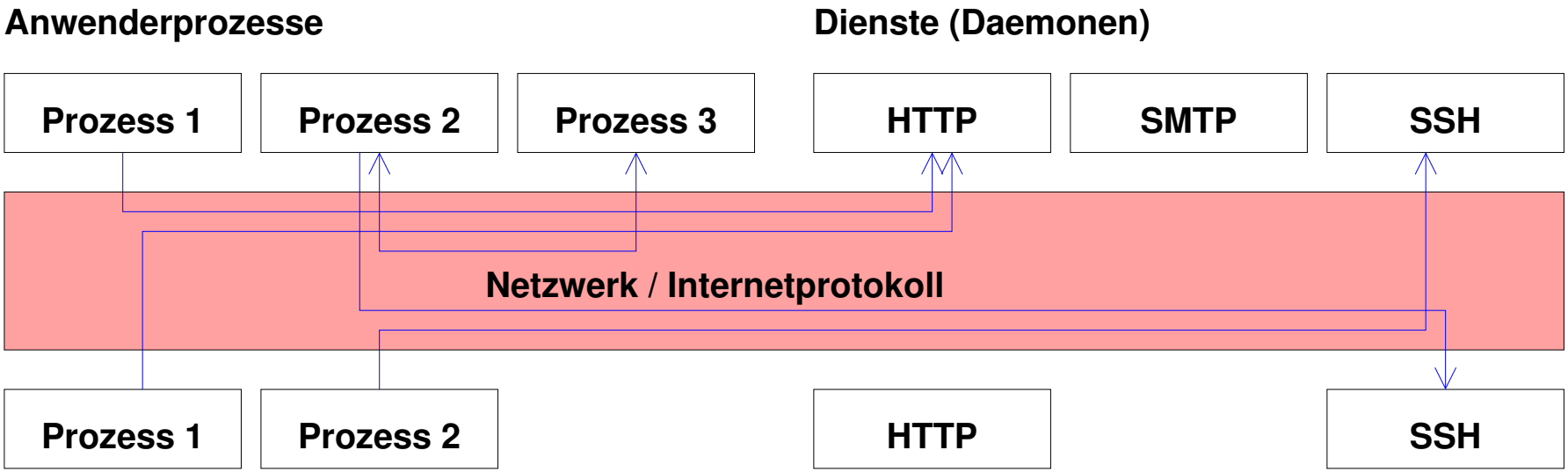
1. `$ echo Hallo Welt | awk '{print $1}'`
   `$ Hallo`
2. `$ echo Hallo Welt | awk '{printf "%s %s!\n",$1,$2}'`
   `$ Hallo Welt!`
3. `$ ps aux > processes` : Schreibt die Liste der Prozesse in eine Datei.
4. `$ awk '{ print }' processes` : zeigt was in der Datei `processes` geschrieben ist.
5. `$ awk '{print $1,$2,$3,$4}' processes` : gibt die ersten 4 Spalten aus.
6. `$ awk '/httpd/ {print $1,$2,$3,$4}' processes` : gibt die ersten 4 Spalten aus nur für die Zeilen die das Muster 'httpd' enthalten.

Konfigurationsdateien

- Konfiguration von UNIX-/GNU-Programmen über Textdateien
- versteckte Dateien im Home-Directory
- oft `.<program>rc` (rc für Resource)
- Beispiele:

`.bashrc` `.profile`	wird von der Shell beim Start/Login gelesen. Enthält Umgebungsvariablen (Pfad), aliases, ... **Achtung, Fehler hierdrin können es unmöglich machen, sich anzumelden!** Immer erst per `source .bashrc` testen
`.ssh/config`	Einstellungen für ssh, z.B. `ForwardX11 yes`
`.vimrc`	Einstellungen für vim, z.B. `syntax on`
`.xsession`	Start des graphischen Interfaces **Bei Fehlern nur noch „failsafe“-Login!**

Netzwerkgrundlagen



- Hier auf das *Internet-Protokoll* beschränken:
- *IP-Adresse* oder DNS-(Nameservice): Name bestimmen Rechner
- Dienstprogramme warten auf *ports* (Anschlüsse) auf Anfragen
- Dienste sind Zahlen zugeordnet die auf *ports* verweisen (http=80, ssh=22,...)

Beispiele von Netzwerkdiensten

Protokoll	Server	Client
http (Hypertext, WWW)	apache, zope, httpd	firefox, konqueror, opera
smtp (Sendmail)	sendmail, postfix	thunderbird, kmail, evolution
pop3	dovecot, courier	
ftp (File transfer)	vsftpd, filezilla	ftp, ncftp, firefox, konqueror
ssh	sshd	ssh, putty
cups (Druckerspooler)	cupsd	lpr, Programme mit Druckdialog
X11 (GUI)	Xorg, XF86	Programme mit GUI
NFS (Network File System)	nfsd, Linux-Kernel	Linux-Kernel

SSH: Secure Shell

`ssh [-X] [<user>@]<host>`	Terminal für Benutzer auf einem anderen Rechner öffnen
`scp <src> <host>:<dst>`	Eine Datei per SSH auf einen anderen Rechner kopieren

- Verschlüsselte Verbindung zu einem anderen Rechner (UNIX Standard; Windows-Client: PuTTY)
- Terminal, Dateitransfer, Umleiten von Netzwerkverkehr
- mit `-X`: Umleiten von X11, GUI-Programme per Netzwerk starten
- CIP-Pool-Gateway:
  Verbindung mit Physik-CIP-Pool:

  `ssh.physcip.uni-stuttgart.de`

  ... Verbindung mit Gateway des ICP:

  `(ssh.icp.uni-stuttgart.de)`

  ... und dann weiter auf einen der Pool-Rechner `(cip0)`

Netz per Kommandozeile: HTTP und Mail

wget: per HTTP Daten laden

```
wget <url>
```

- Lädt eine Webseite über HTTP
- Beispiel: `wget http://www.fftw.org/fftw-3.2.2.tar.gz`

mail: Mails verschicken

```
mail -s <subject> <receipient>
```

- Verschicken einer Mail
- Beispiele:

  ```
  mail -s "Lösung" tutor@icp.uni-stuttgart.de
  ```

- Verschiedene E-mail Programme: kmail, mutt, etc.

Netz per Kommandozeile: FTP

`ftp <host>`

- Zugriff auf ein Filesystem auf einem Server
- Server kann auch anonymen Zugriff erlauben
- Befehle:

`help`	Liste aller Befehle
`passive`	auf passive Transfers umschalten (Firewall!)
`ls, cd`	wie gewohnt, aber auf dem Server
`lcd`	lokales Verzeichnis wechseln
`get <file>`	Datei vom Server holen
`put <file>`	Datei auf den Server schieben

`ping <url>`

- `ping` ist ein Diagnose-Werkzeug, mit dem überprüft werden kann, ob ein bestimmter host(url) in einem IP-Netzwerk erreichbar ist.

Telnet

`telnet <url>`

- Ein verbreitetes Netzwerkprotokoll das alt ist und auf einem zeichenorientierten Datenaustausch über eine TCP-Verbindung basiert.
- `telnet` ersetzt viele Programme von Diensten, die man im Internet nutzen will.
- Es wird auch eine Verbindung zwischen zwei Rechnern aufgebaut, auch wenn diese unterschiedliche Betriebssysteme haben
- Zugriff auf alle Ressourcen, wenn die Berechtigung dazu vorhanden ist
- ABER hat keine Sicherheitsfunktionalitäten (Kenntwörter werden z.B. im Klartext geschickt). Wegen Vollzugriff können Hacker leichtes Spiel haben.

Druckerkontrolle

- `lpstat -a`
  Gibt eine Liste der verfügbare Drucker aus.

- `lpr -P`Druckerwarteschlange(print queue) Dateiname
  Schickt eine Datei zum Drucker sobald dieser frei ist. Jeder Job bekommt eine Job-Nummer.

- `lpq -P`Druckerwarteschlange(print queue)
  Überprüft den Status der angegebenen Druckerwarteschlange.

- `lprm -P`Druckerwarteschlange(print queue) Job-Nummer
  Löscht den jeweiligen Job aus der Druckerwarteschlange.

Grundlegende Netzwerkbefehle

`netstat`	Zeigt aktive Ports
`netcat <host> <port>`	Terminal als Verbindung zu einem Port
`netcat -l -p <port>`	Terminal als Dienst auf einem Port
`hostname`	Wie heißt mein Rechner?
`/sbin/ifconfig`	Infos unter Linux über Netzwerkkarten (IP- und MAC-Adressen!)

Ein paar Tips

- Eine IPv4-Adresse hat die Form a.b.c.d mit $0 \leq$ a,b,c,d ≤ 255
- Eigener Rechner ist stets auch „localhost“ (127.0.0.1)
- Benutzergestartete Dienste nur auf Ports > 1024
- Per NFS können Verzeichnisse auf mehreren Rechnern gleichzeitig zu sehen sein (z.B. in den CIP-Pools)

Letzte Tips

- Im WWW findet sich fast immer eine Lösung
- OpenSuSE-, Ubuntu-, Redhat-Foren
- Fehlermeldungen genau lesen
- Übungsleiter fragen!