4.Übung

Systemsoftware (SYS)

Christian Baun
cray@unix-ag.uni-kl.de

Hochschule Mannheim – Fakultät für Informatik
Institut für Robotik

26.10.2007

Wiederholung vom letzten Mal

- Dateirechte ändern (`chmod`)
- Ändern des Passworts (`passwd`)
- Shell beenden bzw. Benutzer abmelden (`exit`)
- System neu starten oder herunterfahren (`halt`, `reboot`, `shutdown`)
- Benutzeraccounts anlegen (`useradd`)
- Benutzeraccounts löschen (`userdel`)
- Benutzeraccounts ändern (`usermod`)
- Gruppen anzeigen (`groups`)
- Gruppen löschen (`groupdel`)
- Gruppen anlegen (`groupadd`)
- Gruppen ändern (`groupmod`)
- Eigentümer und Gruppenzugehörigkeit ändern (`chown`, `chgrp`)

Heute

- Einführung für Linux/UNIX-Anwender (Teil 3)
  - Zugriffsrechte voreinstellen (`umask`)
  - Hard Links und Symbolische Links (`ln`)
  - Dateien durchsuchen (`grep`)
  - Verzeichnisse packen und entpacken (`zip`, `rar`, `tar`, `gzip`,...)
  - Editoren: Joe's Own Editor, vi(m), Emacs
  - Prozesse anzeigen (`ps`)
  - Prozesse in den Hintergrund schicken (`bg`)
  - Prozesse in den Vordergrund holen (`fg`)
  - Prozesse beenden (`kill`, `killall`)
  - Prozessprioritäten festlegen und ändern (`nice`, `renice`)
  - Prozessvererbung anzeigen (`pstree`)
  - Prozesse/Kommandos verknüpfen mit Pipes (`|`)
  - Wildcards (`?`, `*`, `[]`, `!` und `^`)

Zugriffsrechte voreinstellen – `umask`

- Jede neu erzeugte Datei und jedes neue Verzeichnis wird mit Zugriffsrechten ausgestattet.
- Wie diese voreingestellten Zugriffsrechte aussehen, kann der Benutzer mit dem Kommando `umask` beeinflussen.
- Dem Kommando wird eine Dateierzeugungsmaske übergeben. Diese ist genau wie bei `chmod` eine Oktalzahl.
- Mit der Dateierzeugungsmaske werden die Zugriffsrechte berechnet, die eine neue Datei bzw. ein neues Verzeichnis erhält.
- Wird umask ohne Parameter aufgerufen, wird die aktuelle Dateierzeugungsmaske ausgegeben.

```
$ umask
0022
```

Das Kommando `umask` verstehen

- Wird `umask` mit dem Parameter `-S` aufgerufen, wird die eingestellte Zugriffsberechtigung symbolisch dargestellt, was die Lesbarkeit erhöht.

```
$ umask -S
u=rwx,g=rx,o=rx
```

- Bei `umask` setzt man die Berechtigungen, die Sie nicht automatisch vergeben werden sollen.
- `umask` geht von folgenden Maximalwerten aus:

Neue Dateien	`rw-rw-rw-`	(666)
Neue Verzeichnisse	`rwxrwxrwx`	(777)

- Die Werte in der Dateierzeugungsmaske von `umask` werden von diesen Maximalwerten abgezogen.

Arbeitsweise von `umask`

Maximalwert (Dateien)	`rw-rw-rw-`	(666)
Abzug	`----w--w-`	(022)
Ergebnis	`rw-r--r--`	(644)

- Ist dieses Ergebnis erwünscht, muss `umask` mit der Dateierzeugungsmaske 022 aufgerufen werden:

```
$ umask 022
```

- Für alle neu erzeugten Verzeichnisse berechnen sich die Zugriffsrechte wie folgt:

Maximalwert (Verzeichnisse)	`rwxrwxrwx`	(777)
Abzug	`----w--w-`	(022)
Ergebnis	`rwxr-xr-x`	(755)

Verweise (Links)

- Jede Datei hat im Dateisystem einen **Inode** mit einer eindeutigen Indexknotennummer, mit der das System die Datei identifiziert.
- Ein Inode ist eine Datenstruktur, in der sich u.a. folgende Informationen befinden: Dateigröße, physikalische Position auf der Festplatte, Zugriffsrechte, Erstellungs- und letzte Modifikationszeit.
- Jeder Dateiname ist nur ein Verweis (**Link**) auf eine Indexknotennummer (Inode).
- Jeder Indexknotennummer können beliebig viele Dateinamen als Links zugeordnet werden.
- Jede solche Zuordnung ist ein **Hard-Link** bzw. eine Referenz.

```
$ ls -i folien_bts_uebung2.pdf
249180 folien_bts_uebung2.pdf
```

Hard Links

- **Hard Links** sind zusätzliche Verzeichniseinträge. Wird ein Hard Link erzeugt, wird ein weiterer Eintrag mit einem anderem Namen, auf den gleichen Indexknoten (Inode) vorgenommen.
  $\Longrightarrow$ Hard Links sind Zeiger auf Indexknoten (Inodes)
- Die erste Referenz (Hard Link) auf eine Indexknotennummer wird automatisch erzeugt, wenn die Datei erstellt wird.
- Das Betriebssystem zählt die Referenzen auf eine Indexknotennummer mit und erst, wenn die Anzahl null ist, ist die Datei gelöscht.
- Die Einträge `.` und `..` in jedem Verzeichnis sind Hard Links.
- Hard Links auf eine Indexknotennummer legt das Kommando `ln` an.

```
$ ln <alterDateiname> <neuerDateiname>
```

Symbolische Links

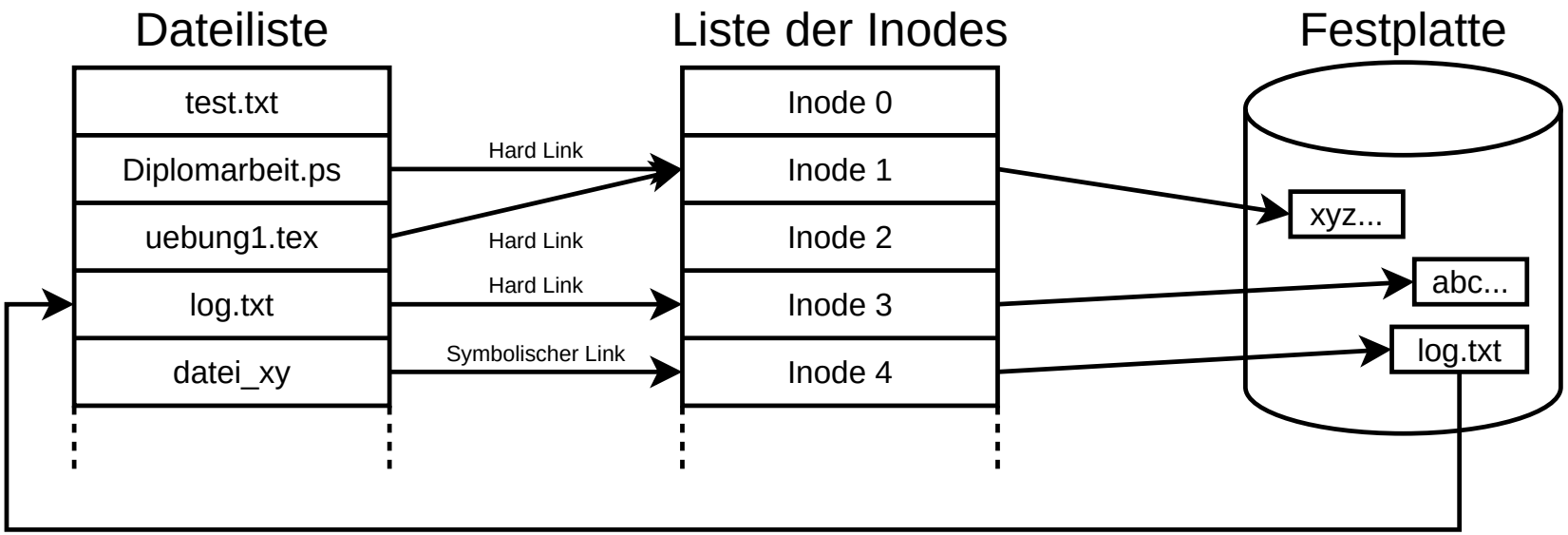
- **Symbolische Links** sind neue Indexknoten mit einem weiteren Namen für eine Datei.
  $\Longrightarrow$ Symbolische Links sind Dateien, die auf andere Dateien zeigen.
- Symbolische Links sind eigentlich Textdateien, die nur den kompletten Pfad zu der gewünschten Datei enthalten.
- Bei Symbolischen Links sind die Indexknotennummern verschieden.
- Mit dem Kommando `ls -l` erkennt man Symbolische Links an dem `l` in der ersten Spalte der Dateiattribute und hinter dem Dateinamen befindet sich ein Pfeil, mit dem Pfad der referenzierten Datei.

```
lrwxrwxrwx 1 user gruppe 14 2006-10-23 23:18 symbLink -> datei
```

- Symbolische Links legt man mit dem Kommando `ln -s` an.

```
$ ln -s <alterDateiname <neuerDateiname>
```

Hard Links und Symbolische Links



- Hard Links funktionieren nur innerhalb eines Dateisystems.
- Symbolische Links funktionieren auch über Dateisystemgrenzen hinweg.
- Alle Links können wie gewöhnliche Dateien mit dem Kommando `rm` entfernt werden.

Dateien durchsuchen – grep

`grep [Option] ... Muster [Datei] ...`

- Das Kommando grep durchsucht Dateien nach einem Muster.
- grep gibt alle Zeilen aus, in denen das Muster enthalten ist.
- Das Muster kann ein regulärer Ausdruck sein.

`-c`	Gibt die Anzahl der Trefferzeilen aus (*count*).
`-i`	Ignoriert die Groß- und Kleinschreibung (*ignore case*).
`-r`	Liest alle Dateien in den Unterverzeichnissen rekursiv (*recursive*).
`-v`	Gibt alle Zeilen aus, die das Suchmuster nicht enthalten (*revert match*).
`-n`	Gibt zu den Zeilen die Zeilennummern aus (*line number*).
`-l`	Gibt nur die Dateinamen mit den Treffern zurück (*files with matches*).
`-L`	Gibt nur die Dateinamen ohne Treffer zurück (*files without matches*).

Einige Beispiele zu `grep`

- Anzahl der Aufzählungen in den Folien von letzter Woche:

```
$ grep -c \begin{itemize} folien_bts_uebung1.tex
```

- Ausführbare Datei des Programms `xlogo` finden:

```
$ locate xlogo | grep bin
```

- Liste der Benutzer, die als Bootshell die `bash` verwenden:

```
$ cat /etc/passwd | grep /bin/bash
root:x:0:0:root:/root:/bin/bash
userx:x:1000:1000:userx:/home/userx:/bin/bash
usery:x:1001:100:usery:/home/usery:/bin/bash
```

- Die Namen aller Dateien im aktuellen Verzeichnis und seinen Unterverzeichnissen ausgeben, die LaTeX-Quellcode enthalten:

```
grep -r \begin{document} *
```

Verzeichnisse packen/entpacken mit `zip`, `bz2` und `rar`

Archiv.zip

packen:	`zip -r Archiv.zip Verzeichnis`
entpacken:	`unzip -x Archiv.zip`
Inhalt anzeigen:	`unzip -l Archiv.zip`

Archiv.rar

packen:	`rar a -r Archiv.rar Verzeichnis`
entpacken:	`rar x Archiv.rar`
Inhalt anzeigen:	`rar l Archiv.rar`

Archiv.arj

packen:	`arj a -r Archiv.arj Verzeichnis`
entpacken:	`arj x Archiv.arj`
Inhalt anzeigen:	`arj l Archiv.arj`

Verzeichnisse packen/entpacken mit `tar`, `gz` und `bz2`

Archiv.tar

packen:	`tar -cvf Archiv.tar Verzeichnis`
entpacken:	`tar -xvf Archiv.tar`
Inhalt anzeigen:	`tar -tvf Archiv.tar`

Archiv.tar.gz

packen:	`tar -cvzf Archiv.tar.gz Verzeichnis`
entpacken:	`tar -xvzf Archiv.tar.gz`
Inhalt anzeigen:	`tar -tvzf Archiv.tar.gz`

Archiv.tar.bz2

packen:	`tar -cvjf Archiv.tar.bz2 Verzeichnis`
entpacken:	`tar -xvjf Archiv.tar.bz2`
Inhalt anzeigen:	`tar -tvjf Archiv.tar.bz2`

Joe's Own Editor – `joe` (1)

Beenden	`Strg-K X`	Datei speichern und Joe beenden
	`Strg-C`	Joe beenden
Speichern	`Strg-K D`	Speichern
	`Strg-K W`	Markierten Block in Datei speichern
Hilfe anzeigen	`Strg-K H`	Hilfe ein-/ausblenden
Block	`Strg-K B`	Anfang eines Blocks
	`Strg-K K`	Ende eines Blocks
Löschen	`Strg-K Y`	Block löschen
	`Strg-Y`	Aktuelle Zeile löschen
	`Strg-D`	Aktuelles Zeichen löschen
	`Strg-W`	Aktuelles Wort löschen
	`Strg-J`	Bis zum Ende der Zeile löschen

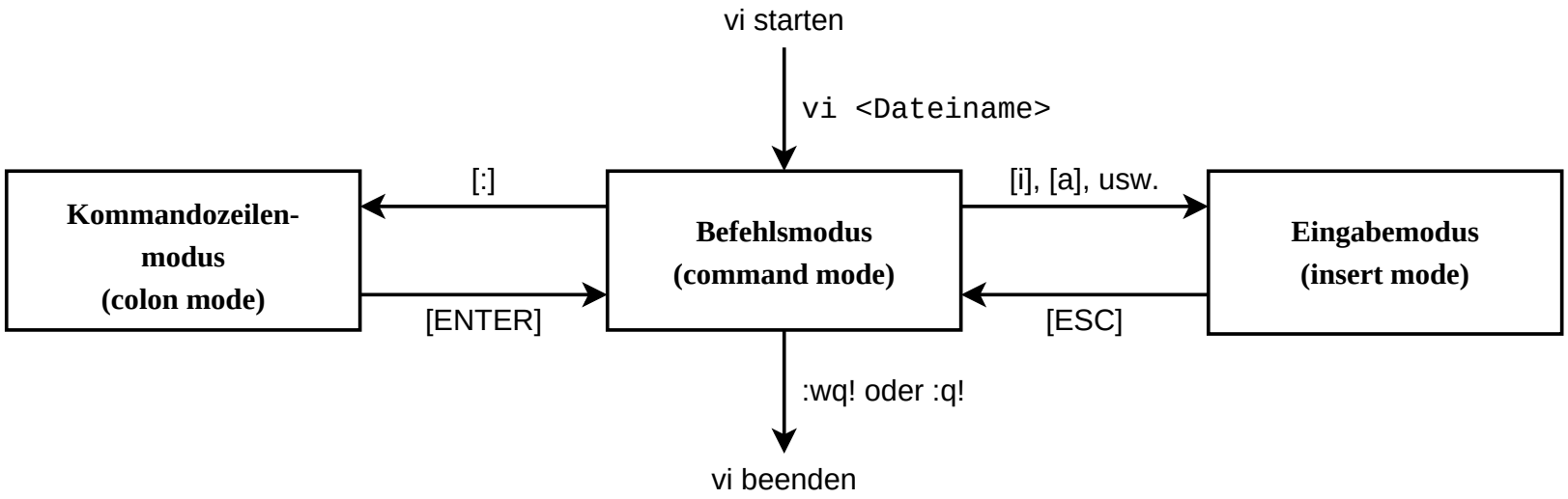
Joe's Own Editor – `joe` (2)

Verschieben	`Strg-K M`	Block verschieben
Kopieren	`Strg-K C`	Block kopieren
Suchen	`Strg-K F`	Suchen
	`Strg-L`	Weitersuchen
Undo/Redo	`Strg-_`	Rückgängig machen (Undo)
	`Strg-Strg`	Wiederherstellen (Redo)
Springen	`Strg-K U`	Zur ersten Zeile der Datei springen
	`Strg-K V`	Zur letzten Zeile der Datei springen
	`Strg-K L`	Zu einer bestimmten Zeile springen
	`Strg-X`	Zum nächsten Wort springen
	`Strg-Z`	Zum vorherigen Wort springen
	`Strg-A`	Zum Anfang der Zeile springen
	`Strg-E`	Zum Ende der Zeile springen

Der Editor `vi`

- vi (*visual*) existiert seit 1976 und ist der Standardeditor unter Unix und auf jedem Linux/UNIX-System zu finden.
  - Vorteile: Minimaler Ressoucenverbrauch, überall verfügbar.
  - Nachteil: Schlechte Benutzbarkeit für Einsteiger.
- Es gibt mehrere Weiterentwicklungen von vi. $\Longrightarrow$ Vim (Vi IMproved).
- vi besitzt drei grundsätzlich Arbeitsmodi:
  - **Befehlsmodus** bzw. Kommandomodus (*command mode*)
  - **Eingabemodus** bzw. Einfügemodus (*insert mode*)
  - **Kommandozeilenmodus** bzw. Komplexbefehlsmodus (*colon mode*)
- Beim Start von vi befindet man sich im Befehlsmodus.
- vi starten: Kommando `vi` oder `vi <Dateiname>`

Zwischen den Arbeitsmodi von `vi` wechseln



- Typische erste Erfahrung mit vi: *Wie komme ich hier wieder heraus?!*

1. ESC-Taste drücken um in den Befehlsmodus zu kommen.
2. `:q!` (*quit*) oder `:wq!` (*write & quit*) eingeben.

Der Eingabemodus von `vi`

- Um in den Eingabemodus zu kommen, gibt es mehrere Kommandos:

`i`	Vor dem Cursor einfügen (*insert*).
`a`	Hinter dem Cursor anhängen (*append*).
`I`	Am Anfang der Zeile einfügen.
`A`	Am Ende der Zeile anhängen.
`o`	Eine neue Zeile unter der aktuellen Zeile einfügen.
`O`	Eine neue Zeile über der aktuellen Zeile einfügen.

- Die ESC-Taste beendet den Einfügemodus.

Der Kommandozeilenmodus von `vi`

- Den Kommandozeilenmodus erreicht man durch einen Doppelpunkt : im Befehlsmodus.
- Durch wechseln in den Kommandozeilenmodus befindet sich der Cursor in der letzten Zeile des Fensters.

`:x`	Beenden und nur bei Änderungen speichern.
`:w`	speichern.
`:q`	beenden.
`:q!`	beenden und Änderungen verwerfen.
`:wq`	Speichern und beenden.
`:w!`	Speichern und Schreibschutz (wenn möglich) ignorieren.
`:w testDatei`	Speichern unter dem Dateinamen `testDatei`.

Der Befehlsmodus von `vi`

- Der Befehlsmodus kennt sehr viele Befehle. Einige wenige:

`x`	Aktuelles Zeichen löschen.
`dd`	Aktuelle Zeile löschen.
`d$`	Von der aktuellen Position bis zum Ende der Zeile löschen.
`dL`	Bis zum Fensterende löschen.
`dG`	Bis zum Dateiende löschen.
`s`	Zeichen ersetzen.
`S`	Ganze Zeile ersetzen.
`C`	Von der aktuellen Position bis zum Ende der Zeile ersetzen.
`/`	Vorwärts suchen.
`?`	Rückwärts suchen.
`n`	Suche fortsetzen.
`n`	Suche in entgegengesetzte Richtung fortsetzen.

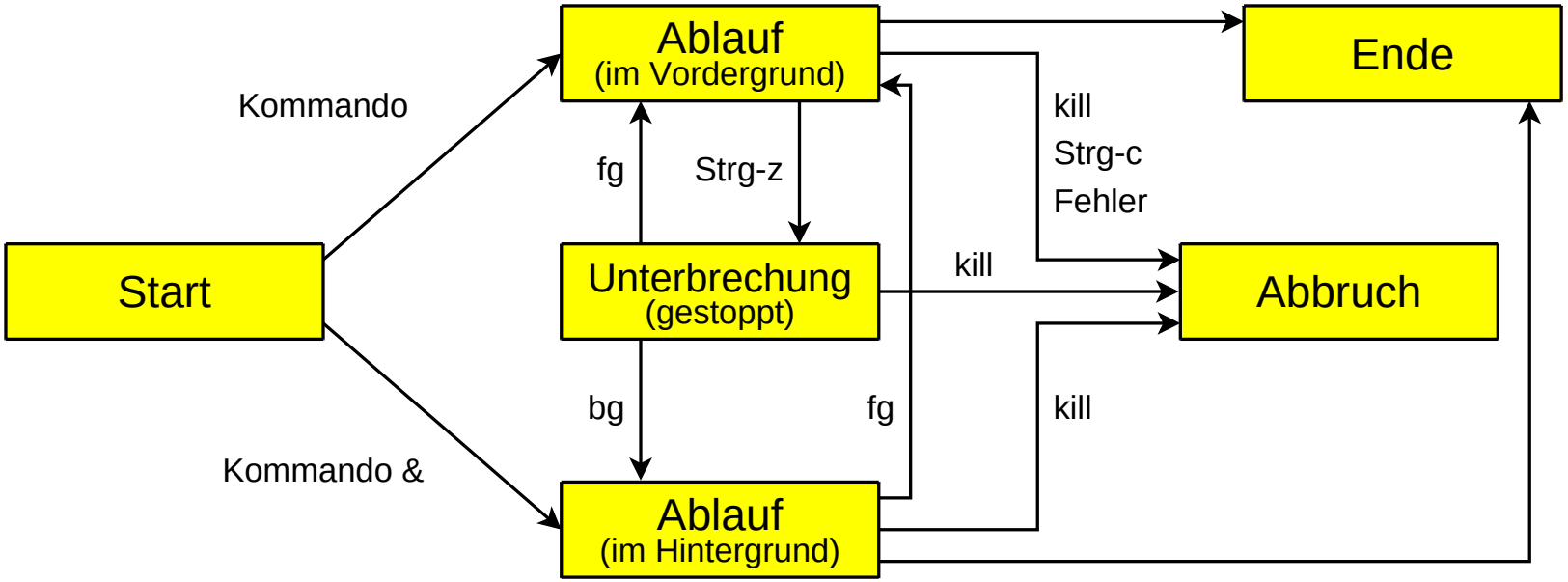
Emacs

- Emacs ist ein sehr mächtiger, über Plugins erweiterbarer Texteditor.
- Ursprünglich von Richard Stallman entwickelt.
- Syntaxhervorhebung (*syntax highlighting*) für die meisten Sprachen.
- Bekannter Fork für das X Window System $\Longrightarrow$ XEmacs
- Es gibt zahlreiche Tools, die in Emacs enthalten sind und nachträglich eingebaut werden können. Dazu gehören WebBrowser, FTP-Client, Spiele, Kalender, Email-Programm, IRC-Client, usw.
- Lange Einarbeitungszeit. Danach ein hilfreiches Werkzeug $\Longrightarrow$ siehe `vi`

Prozesse

- Immer wenn ein neues Programm gestartet wird, wird in Linux/UNIX ein **neuer Prozess** erzeugt.
- Mit Kommandos und Tastenkürzeln können die Benutzer Prozesse steuern.
- Ein Prozess kann sich in unterschiedlichen Zuständen befinden.
- Der erste Prozess eines Systems ist bei unixartigen Betriebssystemen immer der sogenannte **init**-Prozess. init ist der Elternprozess aller Prozesse und hat die Prozess ID 1. Init startet alle anderen Prozesse.
- Prozesse erhalen beim Start eine **Jobnummer** und die **Prozessidentifikation** (PID) zur eindeutigen Identifikation.

Prozessmanipulation (1)



Prozessmanipulation (2)

- Prozess im **Hintergrund** starten: $\Longrightarrow$ an das Kommando & anhängen.

```
$ xclock &
[1] 29593
```

- Prozess im Vordergrund abbrechen: $\Longrightarrow$ `Strg-c`
- Prozess im Vordergrund anhalten/stoppen: $\Longrightarrow$ `Strg-z`
  - Kommando `bg` (*background*) schickt einen Prozess in den Hintergrund.
  - Kommando `fg` (*foreground*) holt einen Prozess in den Vordergrund.
- Laufende Prozesse (Jobs) der aktiven Shell anzeigen lassen: $\Longrightarrow$ `jobs -l`

Prozesse anzeigen – ps

```
ps [Option] ...
```

- Das Kommando ps zeigt eine Momentaufnahme der aktuellen Prozesse.
- **Achtung:** Das Kommando verwendet mehrere Arten von Optionen – mit und ohne einem oder zwei Bindestrichen.

-A oder -e	Wirklich alle Prozesse anzeigen.
-l oder l	Langformat.
-a	Die Prozesse aller Benutzer anzeigen.
-r	Zeigt nur die laufenden Prozesse (STAT: R+).
-t <Nummer>	Zeigt Prozesse des Terminals <Nummer> an.
T	Alle Prozesse für dieses Terminal anzeigen.
f	Prozesshierarchie als Baum anzeigen.
x	Zeigt Prozesse, die von keinem Terminal kontrolliert werden.

Spalten der Ausgabe von `ps`

- **COMMAND**: Name des Kommandos.
- **UID**: Der Benutzername (*User-ID*) des Prozess-Eigentümers.
- **PID**: Die Prozessidentifikation (*Process-ID*).
- **PPID**: Die Prozessidentifikation des Elternprozesses (*Parent Process-ID*).
- **PGID**: Die Prozessgruppe (*Process-Group-ID*).
- **TTY**: Nummer des kontrollierenden Terminals.
- **TIME**: Verbrauchte Rechenzeit.
- **STAT**: Status des Prozesses:
  - **R**: Laufend (*running*)
  - **S**: Schlafend (*sleeping*)
  - **D**: Nicht-störbarer Schlaf (*dormant*)
  - **T**: Angehalten (*stopped*)
  - **Z**: Zombie
- **STIME**: Startzeitpunkt des Prozesses.
- **NICE**: Nice-Wert des Prozesses.

Prozesse in den Hintergrund schicken – bg

`bg [%Jobnummer]`

- Einen Prozess anhand seiner Jobnummer in den Hintergrund schicken:

```
user@server:~$ sleep 1000
^Z
[2]+  Stopped                 sleep 1000
user@server:~$ bg
[2]+ sleep 1000 &
```

- Mit dem Tastenkürzel `Strg-z` wird das Signal 19 (`SIGSTOP`) gesendet.
- Sind mehrere Prozesse im gestoppten Zustand, kann mit `%Jobnummer` festgelegt werden, welcher Prozess im Hintergrund weiterlaufen soll.

Prozesse in den Vordergrund holen – `fg`

`fg [%Jobnummer]`

- Einen Prozess anhand seiner Jobnummer in den Vordergrund holen:

```
user@server:~$ sleep 100 & sleep 500 & sleep 1000 &
[2] 2681
[3] 2682
[4] 2683
user@server:~$ fg %4
sleep 1000
```

- Wird `fg` ohne `%Jobnummer` aufgerufen, holt es den zuletzt im Hintergrund gestarteten Prozess in den Vordergrund.

Was bedeutet + und - bei `jobs`?

```
user@server:~$ jobs -l
[1]  16841 Running                 xclock &
[2]  16842 Running                 xclock &
[3]  16843 Running                 xclock &
[4]- 16844 Running                 xclock &
[5]+ 16845 Running                 xclock &
```

- Der letzte gestartete Prozess ist mit + markiert. Wenn die Kommandos `fg` (*foreground*) und `bg` (*background*) ohne Optionen aufgerufen werden, beziehen sie sich auf den Prozess mit dem +. Dieser Prozess kann auch mit `fg %+` bzw. `bg %+` angesprochen werden.
- Der Prozess mit dem - ist in der Hierarchie direkt nach dem Prozess mit dem +. Er kann direkt mit `fg %-` bzw. `bg %-` angesprochen werden.

Prozesse beenden – `kill`

- Alle Prozesse können mit dem Kommando `kill` abgebrochen werden.

```
$ kill -Signr Prozessidentifikation
```

- Sollen Prozesse anhand ihrer Jobnummer identifiziert und beendet werden, muss vor der Jobnummer ein Prozentzeichen % stehen.

```
$ kill -Signr %Jobnummer
```

Die wichtigsten Signale bei `kill`

- Die wichtigsten Signale:

`kill -1 Prozessnr`	**SIGHUP**	lese Konfigurationsdateien neu (*hangup*)
`kill -2 Prozessnr`	**SIGINT**	unterbrechen (*interrupt*). Vergleichbar mit `Strg-c` auf der Shell.
`kill -3 Prozessnr`	**SIGQUIT**	beenden (*quit*). Fast identisch zu SIGINT. Einziger Unterschied: Es wird eine Core-Datei als Zeichen für einen schwerwiegenden Laufzeitfehler erzeugt.
`kill -9 Prozessnr`	**SIGKILL**	töten (*kill*). Das Programm hat keine Möglichkeit mehr Daten zu speichern oder aufzuräumen (clean up).
`kill -15 Prozessnr`	**SIGTERM**	terminieren (Standardwert). Ein Prozess soll sich beenden.

Liste der möglichen Signale

```
user@server:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

- Alle Prozesse mit einem identischen Namen ändern: $\Longrightarrow$ `killall`

Prozesse anhand ihres Namens beenden – `killall`

`killall [Option] ... Kommandoname ...`

- Im Gegensatz zu `kill`, spezifiziert `killall` die Prozesse über den Namen der Kommandos, die sie ausgeführt haben.
- Mit der Option `-i` wird jedes Senden nachgefragt.

```
user@server:~$ xclock &
[2] 2424
user@server:~$ xclock &
[3] 2425
user@server:~$ xclock &
[4] 2426
user@server:~$ killall -i xclock
xclock(2424) abbrechen? (y/N) y
xclock(2425) abbrechen? (y/N) y
xclock(2426) abbrechen? (y/N) y
[2]   Beendet                 xclock
[3]-  Beendet                 xclock
[4]+  Beendet                 xclock
```

Prozesse mit einer anderen Priorität starten – `nice`

`nice [Option] ...Kommando [Argument] ...`

- Das Kommando `nice` kann einem Kommando oder Programm eine andere als die voreingestellte Priorität zuweisen. Dadurch erhält ein Prozess prozentual mehr oder weniger Rechenzeit zugeteilt:

  $-20 \Longrightarrow$ höchste Priorität

  $20 \Longrightarrow$ geringste Priorität

- Normale Benutzer dürfen die Priorität ihrer Prozesse verringern. Aber nur der Superuser (root) darf die Priorität von Prozessen erhöhen.

```
user@server:~$ nice -n +10 xclock &
[3] 10095
[2]   Beendet                 nice -n +10 xclock
user@server:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  5631  5601  0  75   0 -   959 wait   pts/3    00:00:00 bash
0 S  1000 10095  5631  1  87  10 -  1463 -      pts/3    00:00:00 xclock
```

Prozesspriorität ändern – `renice`

`renice <Priorität> <Prozessidentifikation> ...`

- Mit dem Kommando `renice` kann Prozessen nachträglich eine andere Priorität zugewiesen werden.
- Alternativ kann mit der Option -g auch eine Prozessgruppe und mit der Option -u ein Benutzername angegeben werden.
- Nur der Superuser (root) darf die Priorität von Prozessen erhöhen.

```
user@server:~$ xclock &
[2] 10283
user@server:~$ renice +15 10283
10283: old priority 0, new priority 15
user@server:~$ renice -15 10283
renice: 10283: setpriority: Keine Berechtigung
```

Prozessvererbung anzeigen – `pstree`

`pstree [Option] ...`

- Das Kommando `pstree` gibt einen Baum mit allen aktiven Prozessen aus.
- Die Baumstruktur symbolisiert die Prozessvererbung.
- Deutlich zu sehen ist `init`, der Elternprozess aller Prozesse (PID 1).

```
user@server:~$ pstree
init-+-acpid
     |-artsd
     |-atd
     |-cron
     |-cupsd
...
```

- Sehr hilfreich bei Kontrolle und Fehlersuche $\Longrightarrow$ `pstree -ca`

Übung zu Prozessen (1)

- Starten Sie das Programm `xclock` fünf Mal im Hintergrund.
- Listen Sie die Jobnummern auf.
- Listen Sie die Prozessidentifikationen (PID) auf.
- Lassen Sie sich alle laufenden Prozesse in einer Baumstruktur ausgeben.
- Beenden/Töten Sie die Jobs auf unterschiedliche Arten mit dem Kommando `kill` und beachten Sie die Rückmeldungen des Kommandos:

```
kill -1 <Prozessidentifikationen>
kill -2 <Prozessidentifikationen>
kill -3 <Prozessidentifikationen>
kill -9 <Prozessidentifikationen>
kill -15 <Prozessidentifikationen>
```

Übung zu Prozessen (2)

- Starten Sie das Programm `xclock` im Vordergrund. Anschließend stoppen Sie es und schicken es in den Hintergrund.
- Starten Sie das Programm `xclock` im Hintergrund, holen Sie es in den Vordergrund und beenden Sie es.
- Starten Sie das Programm `xclock` fünf Mal im Hintergrund und beenden es mit einem Kommandoaufruf.
- Frage: Was wird passieren, wenn man **init** mit `kill` abschießt?

Prozesse verknüpfen mit Pipes – |

- Prozesse haben eine Eingabe und eine Ausgabe.
- Eine Pipe sorgt dafür, dass die Ausgabe eines Prozesses in die Eingabe eines anderen gelangt und wird mit dem senkrechten Balken erzeugt.
- Typisches Einsatzgebiet: Verarbeitung von Texten.

```
# tail -f /var/log/messages | grep eth0
Sep 27 18:21:54 localhost kernel: e1000: eth0: e1000_probe:
Intel(R) PRO/1000 Network Connection
```

```
$ cat folien_bts_uebung3.tex | grep itemize | grep -n begin
1:\begin{itemize}
3:\begin{itemize}
4:\begin{itemize}
...
```

Wildcards – einige der Wichtigsten

`?`	Ersetzt ein beliebiges Zeichen.
`*`	Ersetzt beliebig viele oder Null Zeichen.
`abc*`	Beginnt mit `abc`, danach beliebig viele Zeichen.
`.*`	Das erste Zeichen muss ein Punkt sein. Was danach kommt, ist egal.
`*abc*`	Es muss mindestens die Zeichenfolge abc enthalten sein. Egal wo.
`abc*xyz`	Fängt mit `abc` an und hört mit `xyz` auf.
`[xyz]`	Platzhalter für eines der drei Zeichen in den eckigen Klammern.
`[abc]de`	Kann sein: `ade`, `bde` oder `cde`.
`[a-z]`	Genau ein Zeichen aus dem Bereich der Kleinbuchstaben `a` bis `z`.
`[a-z]x`	Kann sein: `ax`, `bx`, `cx`, `dx` ... `zx`.
`[A-Za-z]xyz`	Kann sein: `Axyz` ... `Zxyz`, `axyz` ... `zxyz`.
`[!x]`	Nicht `x` an dieser Stelle.
`[^x]`	Nicht `x` an dieser Stelle.
`[!abc]`	Nicht `a`, `b` oder `c` an dieser Stelle.

Nächste Übung:

2.11.2007