



Exkurs: ACLs



- Oft feinere / flexiblere Regelung der Zugriffsrechte gewünscht
- ACLs = access control lists
- Features:
 - Individuelle Permissions pro User möglich
 - Selbst-definierte Gruppen
 - Permissions pro selbst-definierter Gruppe
 - ...
- Für rel. kleine Arbeitsgruppen ist das "normale" Unix-Permissions-Modell völlig ausreichend



Weitere File-Attribute

- Zeiten:
 - Modification (write): `ls -l`
 - Creation: `ls -lc`
 - Access (read): `ls -lu`

```
Terminal
Window Edit Options Help
/home/rob% ls -l file
-rw-r----- 1 rob student 343 Dec 5 13:51 file
```

- Größe, Links, ...

mod time



Prozesse

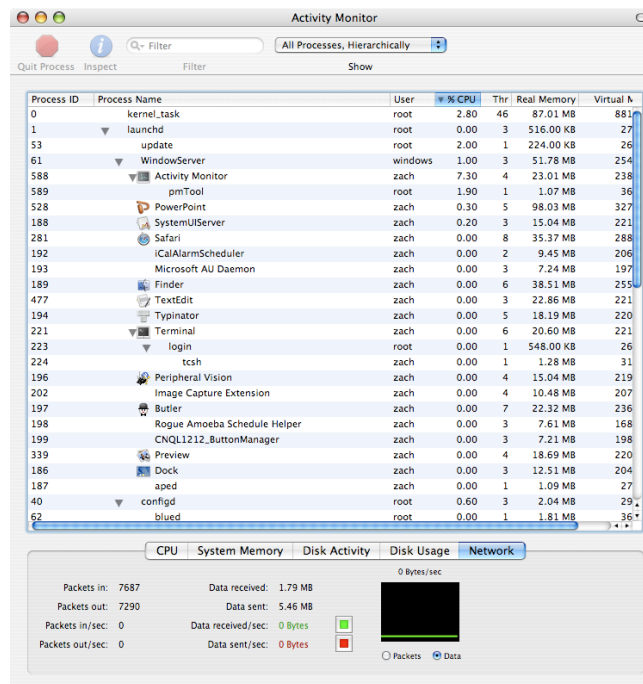


- Programm, das gerade läuft, schläft, oder hängt
- Jeder Prozeß führt ein *Environment* mit sich:
 - Prozeß-ID (PID), User-ID (UID), Group-ID (GID), u.a. IDs
 - current working directory (CWD, manchmal auch PWD)
 - Environment-Variablen (Paare von Strings)
 - ...
- Relative Pfade werden bzgl. des CWD's des Prozesses interpretiert



Spawning processes

- Ein Prozeß wurde *immer* von einem anderen erzeugt
- Heißt Vater-Prozeß (*parent process*)
- Vorgang heißt engl. *to spawn*
- *Child process* erbt das komplette Environment (außer seinen IDs u.ä.)



```
xterm-color --top
Processes: 78 total, 2 running, 76 sleeping, 295 threads 09:53:36
Load Avg: 0.34, 0.43, 0.52 CPU usage: 18.9% user, 19.4% sys, 69.8% idle
SharedLibs: num = 226, resident = 35.7M code, 3.77M data, 19.5M LinkEdit
MemRegions: num = 20539, resident = 615M + 10.9M private, 160M shared
PhysMem: 129M wired, 559M active, 230M inactive, 982M used, 41.6M free
VM: 11.1G + 145M 741154(9) pageins, 654697(0) pageouts

PID COMMAND %CPU TIME #TH #PRTS #RREGS #RVRTS RSHRD RSIZE VSIZE
7566 screencapt 0.0% 0:00.03 1 36 33 364K 788K 1.27M 155M
7562 top 20.9% 0:01.68 1 19 22 864K 364K 1.27M 26.9M
7556 mdimport 0.0% 0:00.32 3 61 47 880K 2.80M 2.48M 39.0M
7546 mdimport 0.0% 0:05.95 4 65 155 13.4M 4.19M 15.1M 68.7M
7534 Finder 0.0% 0:36.25 4 125 307 6.71M 40.1M 45.6M 266M
7498 espoclient 0.0% 0:00.66 2 35 31 780K 676K 1.45M 27.6M
7492 lookupd 0.0% 0:00.51 2 34 37 480K 852K 1.30M 28.9M
6934 ATSServer 0.0% 0:14.35 2 147 261 1.80M 16.7M 1.75M 94.2M
6912 tail 0.0% 0:00.00 1 13 18 0K 276K 204K 26.7M
6897 tail 0.0% 0:00.00 1 13 18 0K 276K 204K 26.7M
6670 tail 0.0% 0:00.00 1 13 18 4K 276K 204K 26.7M
5996 tcsh 0.0% 0:00.70 1 14 23 600K 604K 980K 31.2M
5995 login 0.0% 0:00.02 1 16 37 0K 360K 288K 26.9M
5415 iTunesHelp 0.0% 0:00.50 2 59 161 348K 6.47M 8.02M 207M
4644 Microsoft 0.0% 0:28.16 4 93 191 936K 6.73M 6.73M 208M
4643 Microsoft 0.3% 50:17.03 10 148 2208 346M 75.2M 167M 968M
4495 Preview 0.0% 2:29.22 5 121 683 15.6M 21.7M 25.5M 278M
4266 AppleSpell 0.0% 0:00.76 1 33 35 372K 2.10M 1.07M 37.3M
2152 Safari 0.0% 45:39.67 11 303 2290 121M 31.9M 108M 667M
935 capsd 0.0% 1:13.89 2 30 29 3.27M 848K 3.91M 30.9M
605 System Eve 0.0% 0:05.20 2 70 203 860K 6.78M 4.20M 209M
480 Dashboard 0.0% 0:06.09 6 116 260 2.12M 9.98M 11.1M 217M
479 Dashboard 0.6% 15:40.77 6 117 316 5.39M 9.75M 14.1M 226M
478 Dashboard 1.5% 53:20.47 6 115 289 1.76M 13.7M 10.3M 231M
477 Dashboard 0.0% 0:02.30 5 91 253 1.62M 12.6M 8.80M 221M
476 Dashboard 0.0% 0:05.89 6 123 337 2.44M 11.9M 10.9M 224M
475 Dashboard 0.0% 0:02.12 5 106 250 1.15M 12.1M 7.50M 221M
474 Dashboard 0.0% 0:28.86 6 258 494 11.8M 13.7M 19.5M 231M
473 Dashboard 0.0% 0:04.13 6 116 219 1.58M 9.14M 7.41M 211M
472 Dashboard 0.0% 2:15.15 9 248 460 4.22M 16.3M 13.6M 250M
471 Dashboard 0.0% 0:17.13 6 126 258 3.46M 6.74M 13.1M 218M
470 Dashboard 0.0% 0:04.17 5 96 256 2.69M 5.95M 12.2M 215M
455 tail 0.0% 0:00.00 1 13 18 4K 276K 172K 26.7M
289 Terminal 0.6% 10:38.32 10 197 290 3.17M 13.2M 13.7M 229M
272 Image Capt 0.0% 0:05.86 4 103 192 736K 6.82M 4.47M 207M
271 UniversaliA 0.0% 7:30.46 2 73 194 800K 6.23M 5.27M 207M
269 CNQL1212_B 0.0% 2:02.87 3 71 173 444K 5.43M 3.39M 199M
268 Rogue Amoe 0.0% 0:01.35 3 65 72 516K 4.01M 4.19M 159M
267 Butler 0.0% 3:18.34 4 106 323 2.76M 14.3M 6.68M 241M
266 Peripheral 0.0% 0:15.84 4 225 320 3.14M 14.3M 8.50M 230M
265 Typinator 0.1% 4:12.30 5 226 277 4.86M 14.0M 10.1M 225M
264 Microsoft 0.0% 0:01.96 3 71 160 456K 4.44M 3.23M 197M
```

top in der Shell



Prozesse aus Sicht der Shell



- 3 Zustände eines Prozesses (aus Sicht der Shell)
 - Foreground: Default
 - Ausgabe (stdout) des Prozesses erscheint im Terminal-Fenster
 - Eingabe (stdin) des Prozesses kommt vom Keyboard
 - Background:
 - Ausgabe erscheint im Fenster
 - Eingabe nicht erlaubt
 - Gestoppt:
 - Prozeß schläft



Kommandos zur Prozesskontrolle



Befehl	Funktion
<code>ps</code>	Prozesse anzeigen
<code>ps -edfjw</code>	Alle Prozesse anzeigen
<code>ps -auxw</code>	dito für einige andere Unix-Varianten
<code>kill pid</code>	Prozeß mit PID <code>pid</code> abbrechen (wie Ctrl-C)
<code>kill -9 pid</code>	... wenn der Prozeß trotzdem nicht aufhören will ☺
<code>command ... &</code>	Prozeß im Hintergrund starten
<code>jobs</code>	Prozesse im Hintergrund anzeigen
<code>Ctrl-C</code>	Foreground-Prozeß abbrechen (interrupt)
<code>Ctrl-Z</code>	Foreground-Prozeß anhalten (stoppen)
<code>fg</code>	Zuletzt angehaltenen Prozeß im Foreground weiterlaufen lassen
<code>bg</code>	Angehaltenen Prozeß im Background weiterlaufen lassen
<code>Ctrl-S</code>	Ausgabe des Foreground-Prozesses anhalten (Pr. läuft weiter!)
<code>Ctrl-Q</code>	Ausgabe weiterlaufen lassen
<code>top</code>	tabellarische Ansicht aller Prozesse und deren CPU-Verbrauch



Bash vs. tcsh



- Annahme im folgenden: **tcsh** (csh) !
 - Sehr häufig ist auch bash, aber IMHO für interaktive CL-Nutzung nicht so komfortabel
- Wie stellt man fest, welche Shell man gerade hat?
 - Einfach 'set' eingeben:

Resultat unter tcsh

```
% set
...
tcsh      6.14.00
...
version  tcsh 6.14.00 ...
```

Resultat unter bash

```
% set
...
BASH=/bin/bash
...
BASH_VERSINFO=( [0]="3" ...
BASH_VERSION='3.2.17(1)-...
```

- Wer bash hat: das Kommando **tcsh** eingeben



Shells: Was passiert mit einem Kommando?



- Shell durchläuft folgenden Zyklus mit jedem eingegebenen Befehl:
 1. Kommando wird aus der Eingabezeile oder einem Script gelesen
 2. Aliases werden expandiert
 3. Variable Substitutions werden vorgenommen
 4. Wildcards (File-Patterns) werden expandiert
 5. Das Kommando wird ausgeführt:
 - Entweder von der Shell selbst (built-in commands)
 - Oder in einem neuen Prozeß



1. Aliases



- Ersatz für häufige / längliche Kommandos
- Beispiel:

```
% alias mo less  
% alias uni "ssh -l zach -X hera.uni-bonn.de"  
% mo program.cpp  
% uni
```

- **alias** zeigt alle Aliases an

Alias 'uni' wird ersetzt ...

Alias 'mo' wird ersetzt ...

Definiert Alias 'uni'

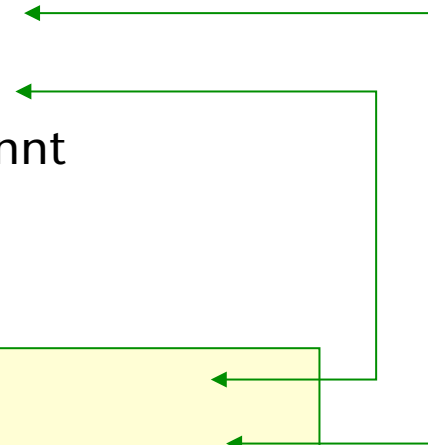
Definiert Alias 'mo'



2. Variablen

- Variable = Name + Wert, Wert = Zeichenkette
- 2 Arten:
 - Normale Shell-Variablen
 - Environment-Variablen,
sind auch in Kind-Prozessen (child process) bekannt
- Setzen:

```
% setenv TMP "/tmp"  
% set tmpdir = "/home/stud/zach/tmp"
```



- Verwenden:

```
% cp file ${tmpdir}  
% echo ${TMP}
```



- Anhängen:

```
% setenv PATH ${PATH}:${HOME}/bin  
% setenv PATH ${HOME}/bin:${PATH}
```



Wichtige Environment-Variablen



Variable	Bedeutung
DISPLAY	Display, auf dem neue Fenster geöffnet werden (Bsp.: aurikel:0.0)
HOME	Home-Verzeichnis
PRINTER	Default-Drucker
TMP	Verzeichnis für temporäre Files
PATH	Suchpfad für Kommandos
MANPATH	Suchpfad für Man-Pages
PWD	Aktuelles Verzeichnis, in dem man sich gerade befindet (CWD)

- **printenv** druckt alle Environment-Variablen aus
- **echo \$VAR** druckt Wert dieser Environment-Variablen



4. File Patterns

- File-Name mit Wildcards: * ? []

Wildcard	Bedeutung
?	Genau ein beliebiges Zeichen
*	Beliebig viele beliebige Zeichen (auch 0)
{0,1,2}	Genau ein Zeichen aus der Menge {0,1,2}
[0-9]	Genau ein Zeichen aus der Menge {0,...,9}
[a-zA-Z0-9]	Genau ein Zeichen aus der Menge {a,...,z,A,...,Z,0,...,9}
[^0-9]	Genau ein Zeichen <i>nicht</i> aus der Menge {0,...,9}

- Beispiele:

```
% ls *.cpp *.h
```

```
% ls [0-9][0-9]*.ppt
```

```
% ls *[^a-zA-Z0-9_.,-]*
```



Kleine Warnung zu rm

Task: Shoot Yourself in The Foot

The proliferation of modern programming languages (all of which seem to have stolen countless features from one another) sometimes makes it difficult to remember what language you're currently using. This handy reference is offered as a public service to help programmers who find themselves in such a dilemma.

```
% ls  
foot.c foot.h foot.o toe.c toe.o  
% rm * .o
```

```
rm: .o no such file or directory  
% ls  
%
```





Quotation



- Achtung: das Kommando sieht die Wildcards nie!
- Shell expandiert Wildcards
- Quotation verhindert, daß Wildcards (allg. Meta-Zeichen) von der Shell expandiert werden
- Arten:
 - \ verhindert Expansion des folgenden Zeichens

```
% echo \*.ppt \ $PATH
```

- "... " verhindert Expansion der Wildcards, erlaubt Variablen

```
% echo "*.ppt $PATH"
```

- '...' verhindert jegliche Expansion (Wildcards, Variablen, ...)

```
% echo '*.ppt $PATH'
```



5. Wie die Shell ein Kommando ausführt

1. Built-in: Shell führt Kommando selbst aus
 - Beispiel: **echo**
2. Sonst: externes Programm
 - Beispiel: **ls dir**
3. **command** in **PATH** (Environment-Variable) suchen
4. Falls nicht gefunden, Fehlermeldung
5. Kind-Prozeß erzeugen
 - Erinnerung: erbt Environment des Vater-Prozesses (Shell)
6. Argumente (Zeichenketten) dem Prozeß bereitstellen
 - Argumente werden durch Space (i.a.) getrennt
7. Warten bis Kind-Prozeß beendet





Suchpfade



- Environment-Variable mit Liste von durch `:` getrennten Verzeichnissen
- Beispiel:

```
% echo $PATH  
./usr/bin:/bin:/usr/local/bin:/home/ll/zach
```

- Kommandos werden in **PATH** gesucht
 - File mit Namen des Kommandos im ersten Verzeichnis und executable?
→ ausführen
 - Sonst: nächstes Verzeichnis in PATH untersuchen ...
- Analog für Man-Pages und andere



File- "Viewer"



- **more, less**
 - Interaktiv, zeigen *jeden* File-Typ im Terminal-Fenster an
 - Unter Linux/Mac ist **more = less**
 - Suchen mit '/', weitersuchen mit 'n', rückwärts 'N', u.v.m.
 - Environment-Variable: **setenv PAGER 'less -l'**
- Nicht-interaktiv:
 - **cat** ("concatenate" = aneinanderhängen)
 - **head** , **tail** = ersten / letzten paar Zeilen anzeigen



Editoren



- Programmierer schreiben ASCII, insbesondere Software
 - Nur für reines ASCII (kein "Markup" irgendwelcher Art)
 - Für kleine Listen
 - Zum Editieren irgendwelcher Text-Files
 - Z.B.: VisualStudio-Project-Files, XML-Files, HTML-Files
 - Vor allem zum "remote" Editieren
- Heiliger Krieg, welches der beste ist
- Ein Programmier-Editor sollte ...
 - Effizientes UI haben (i.A. nicht intuitiv!)
 - Wenige Tasten / Mauskilometer für die häufigen Aktionen
 - Syntax Highlighting
 - Makros
 - Reguläre Ausdrücke zum Suchen und Ersetzen
 - Cross-platform sein



Einige Editoren zur Auswahl

- **vim** / **gvim** (Obermenge von vi, welcher immer installiert ist)
 - Effizientester Editor, steilste Lernkurve
 - Die Homepage von vim: www.vim.org
- **emacs/xemacs** (extrem umfangreich)
 - "Emacs wäre gar kein so schlechtes Betriebssystem, wenn es nur einen brauchbaren Editor hätte" 😊
- **kate** oder **nedit** (kein non-GUI-Mode → taugt nicht zum remote Editieren, sonst durchaus eine Alternative)
- ...



- Meine Empfehlung:
 - Für diejenigen, die sog. "power user" werden wollen: suchen Sie sich einen mächtigen, effizienten, und cross-plattform ASCII-Editor aus, und lernen Sie diesen beherrschen (dauert eine Weile)
 - **vim**, **emacs** / **xemacs**, **nedit**, ...
 - Für alle anderen: nehmen Sie den im Menü angebotenen Editor
 - Unter Linux: **kedit myfile.txt** (oder vom "K"-Menü aus)
 - Falls Sie remote editieren möchten / müssen: **mcedit**
- Reference-Cards für VIM und Emacs auf der Homepage der Vorlesung



Die wichtigsten Befehle in vi / vim / gvim

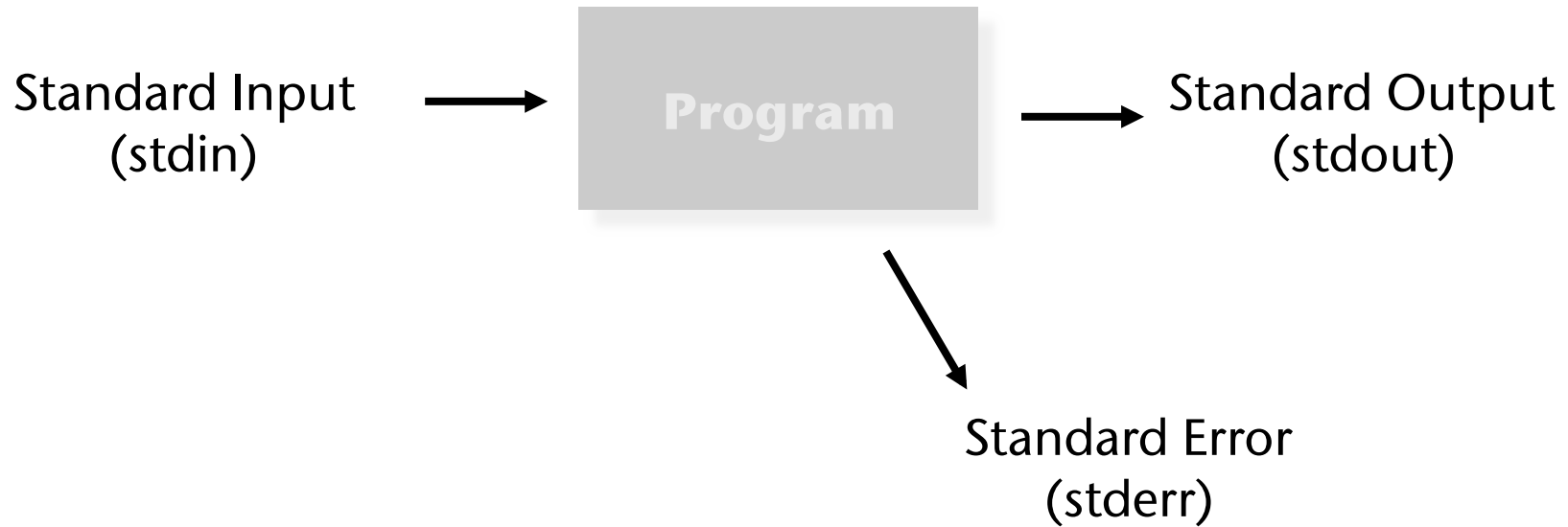
- Besonderheit: Vim ist Mode-basiert !!
- Default- ("Home"-) Modus = Kommando-Modus
 - Aus jedem anderen Mode kommt man mit <Esc> dorthin zurück
- Tasten:
 - 'i' → Insert-Mode = Einfügen von Text an der aktuellen Cursor-Pos.
 - 'R' → Replace-Mode = Überschreiben
 - 'x' → aktuelles Zeichen löschen
 - :w → File speichern
 - :q → Vim verlassen
 - '/' → suchen
 - 'n' → nächstes Vorkommen suchen
 - '.' → letztes Kommando wiederholen





Standard-I/O

- Shell etabliert 3 Kanäle zu / vom Prozeß:



- Default-mäßig mit Terminal (Fenster & Keyboard) verbunden



Redirection

- Verbindet Standard-I/O mit Files
- Prozeß (Programm) bemerkt davon nichts!

Redirection	Bedeutung
> file	stdout wird nach file geschrieben
>> file	stdout wird an file angehängt
>& file	stdout und stderr umlenken
< file	Programm liest aus file, nicht von Keyboard
...	Shell bietet noch viele weitere Möglichkeiten

- Beispiel:

```
% ls -l > dir-listing  
% sort dir-listing
```

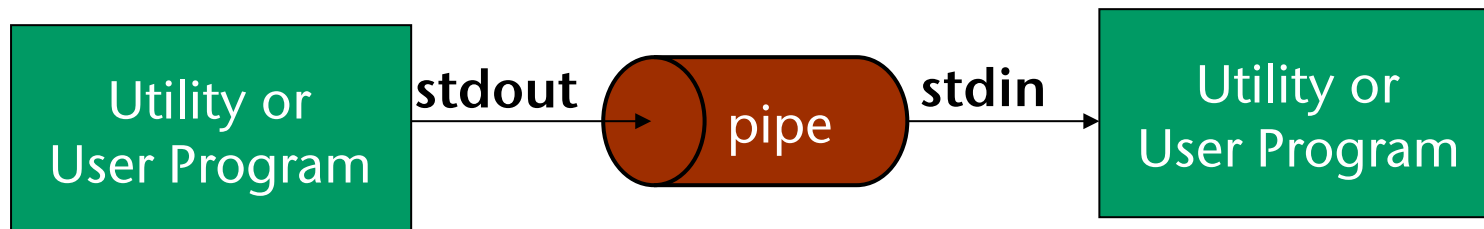
```
% ps -edfyl > procs.log
```




Pipelines



- Effiziente Möglichkeit, komplexere Kommandos aus einfacheren zusammenzubauen!
- Selber Mechanismus wie bei Redirection, um Prozesse miteinander zu verbinden:



- Syntax:

```
% command | command | command | ...
```



Beispiele

- Listings weiterverarbeiten:

```
% ls | sort | more  
% ls -l *.cpp | wc -l  
% ls -l *.cpp | sort > sorted-dir
```

```
% ls -l | tr -s " _"
```

Tip:
**Bauen Sie eine Pipeline immer
eine Stufe nach der anderen auf!!**



Unix-Philosophie

- "Small is beautiful"
- Make each tool do *one* thing only
- Make it do it well
- Read from stdin, write to stdout (if sensible)
- Use ASCII-Files





Filter



- Sind hauptsächlich dazu gedacht, um damit Pipelines aufzubauen
- Generelle Verwendung: Filter lesen von stdin, schreiben auf stdout
- Arbeiten oft zeilenweise

Utility	Funktion
<code>cut</code>	Felder oder Zeichenspalten ausschneiden
<code>fmt</code>	Auf 72 Zeichen umformatieren
<code>sort</code>	Zeilenweise sortieren (auch nach Teil-Key)
<code>uniq</code>	Duplikate entfernen
<code>wc</code>	Zeichen, Wörter, und Zeilen zählen
<code>tr</code>	Zeichen ersetzen
<code>grep</code>	Zeichenketten in der Eingabe suchen (s.u.)
<code>rev</code>	Reihenfolge der Zeichen umkehren (zeilenweise)



Beispiel

- Alle Filegrößen in einen File ausgeben:

```
% ls -l | tr -s " " | cut -d' ' -f5 > sizes.txt
```

- Nur die letzten paar Zeilen eines langen Directory-Listings:

```
% ls -l | tail -10
```