

# Vorkurs – Teil 2

FSI Informatik

Uni Erlangen-Nürnberg

25. Oktober 2014

- 1 Prozessverwaltung
- 2 Pipes
- 3 Rechteverwaltung
- 4 Secure Shell – Fernsteuern anderer Rechner
- 5 Kommandos, Kommandos, Kommandos, ...

Ein gestartetes Programm, das sich in Ausführung befindet, nennt man *Prozess*.

- Jeder Prozess wird über eine systemweit eindeutige numerische ID identifiziert.
- Startet man ein Programm in einem Terminal, nimmt dieses erst dann wieder Befehle entgegen, wenn sich der Prozess beendet hat – der Prozess läuft *im Vordergrund*.

# Prozessverwaltung

Ctrl-C – Prozess abschließen

```
$ cp datei.blob /media/usb/
```

# Prozessverwaltung

Ctrl-C – Prozess abschießen

```
$ cp datei.blob /media/usb/
```

Das Kopieren dauert mir zu lange...

<Ctrl-C>:

^C

```
$ _
```

## Ctrl-C

- **Beendet** den Prozess, der gerade im Vordergrund läuft.
- Gibt die Befehlszeile für weiteres Arbeiten frei.

# Prozessverwaltung

Ctrl-Z – Prozess anhalten

```
$ cp datei.blob /media/usb/
```

# Prozessverwaltung

Ctrl-Z – Prozess anhalten

```
$ cp datei.blob /media/usb/
```

5 Minuten später...

Ein genervtes <Ctrl-Z>:

```
^Z
```

```
[1]+  Stopped
```

```
cp datei.blob /media/usb/
```

```
$ _
```

## Ctrl-Z

- **Pausiert** den Prozess, der gerade im Vordergrund läuft.
- Gibt die Befehlszeile für weiteres Arbeiten frei.

# Prozessverwaltung

fg – Angehaltenen Prozess im Vordergrund fortsetzen

Was jetzt?

Man kann wieder Befehle eingeben, aber der cp-Prozess ist eingefroren und arbeitet nicht weiter.

- Wie komme ich wieder an cp ran?
- Wie lasse ich es weiterlaufen?

# Prozessverwaltung

fg – Angehaltenen Prozess im Vordergrund fortsetzen

Was jetzt?

Man kann wieder Befehle eingeben, aber der cp-Prozess ist eingefroren und arbeitet nicht weiter.

- Wie komme ich wieder an cp ran?
- Wie lasse ich es weiterlaufen?

```
$ fg
cp datei.blob /media/usb/
...nach 10 Minuten...
$ _
```

Das ermöglicht aber noch kein echtes Multitasking, denn im Terminal ist nie mehr als ein Prozess gleichzeitig aktiv.

# Prozessverwaltung

bg – Angehaltenen Prozess im Hintergrund fortsetzen

```
$ cp datei.blob /media/usb/  
<Ctrl-Z>  
^Z  
[1]+  Stopped                  cp datei.blob /media/usb/  
$ bg  
[1]+ cp datei.blob /media/usb/ &  
$ _
```

Hier kann jetzt normal weiter gearbeitet werden!

# Prozessverwaltung

bg – Angehaltenen Prozess im Hintergrund fortsetzen

```
$ cp datei.blob /media/usb/  
<Ctrl-Z>  
^Z  
[1]+  Stopped                  cp datei.blob /media/usb/  
$ bg  
[1]+ cp datei.blob /media/usb/ &  
$ _
```

Hier kann jetzt normal weiter gearbeitet werden!

bg

- Setzt die Ausführung des zuvor unterbrochenen Prozesses im Hintergrund fort, so dass man im Vordergrund sofort weiterarbeiten kann.
- Ermöglicht echtes Multitasking!

# Prozessverwaltung

Programm direkt als Hintergrundprozess starten

```
$ cp datei.blob /media/usb/ &
```

```
$ _
```

Hier kann jetzt normal weiter gearbeitet werden!

# Prozessverwaltung

Programm direkt als Hintergrundprozess starten

```
$ cp datei.blob /media/usb/ &  
$ _
```

Hier kann jetzt normal weiter gearbeitet werden!

... und nach 10 Minuten:

```
$ cd ~  
[1]+  Done                cp datei.blob /media/usb/  
$ _
```

## & am Ende eines Befehls

- Führt den eingegebenen Befehl aus und...
- ... gibt das Terminal sofort wieder frei.

# Prozessverwaltung

jobs – Hintergrundprozesse anzeigen

```
$ jobs
[1]-  Stopped      sleep 1
[2]   Running     sleep 100 &
[3]+  Stopped      cat

$ fg %1
sleep 1
```

## fg und bg

- Mit `%n` auf einen Job anwenden.
- Sonst wird der Job mit dem `+` angenommen (hier `%3`).

# Prozessverwaltung

## kill – Beenden von Prozessen

Mit Ctrl-C kann man nur den aktuell laufenden Vordergrundprozess töten. Aber wie werde ich einen Prozess los, der nicht im Vordergrund läuft?

```
$ jobs
[1]-  Stopped      sleep 2
[2]+  Running      sleep 1d &
```

# Prozessverwaltung

## kill – Beenden von Prozessen

Mit Ctrl-C kann man nur den aktuell laufenden Vordergrundprozess töten. Aber wie werde ich einen Prozess los, der nicht im Vordergrund läuft?

```
$ jobs
[1]-  Stopped                  sleep 2
[2]+  Running                  sleep 1d &

$ kill %2

$ cd ~
[2]+  Terminated             sleep 1d &
```

# Prozessverwaltung

ps – Prozesse auflisten

kill %n:

- Funktioniert nur bei Prozessen, die man in der aktuellen Shell gestartet hat.
- ... aber ich hätte gerne auch die Möglichkeit meinen Browser abzuschießen, den ich außerhalb der Shell gestartet habe!

kill kann auch das, benötigt aber die entsprechende Prozess-ID (PID). Die bekommt man über das Programm ps.

ps

Wichtige Parameter:

- u zeigt ausführliche Informationen über Prozesse
- x zeigt auch Prozesse, die nicht an ein Terminal gebunden sind

# Prozessverwaltung

ps – Prozesse auflisten

```
$ ps ux
```

```
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
simagrop 14752  0.0  0.2   6700   2140 pts/2    Ss+  15:47   0:00 bash
simagrop 16744  1.1  6.1 227220 62700 ?        Ssl  16:06   2:10 /usr/lib/iceweasel/firefox-bin
```

## Wichtige Spalten

**PID** Prozess-ID

**%CPU** CPU-Auslastung

**RSS** Speicherbedarf im RAM

**STAT** Aktueller Prozesszustand (siehe *manpage*)

**TIME** Rechenzeit, die der Prozess effektiv verbraten hat

Wenn man sich primär für die Systemlast interessiert, verwendet man an Stelle von `ps` entweder `htop` (in bunt!) oder `top`.

Mit der PID von vorher:

```
$ kill 16744
```

- Falls `kill` nichts ausgibt, ist das meist ein gutes Zeichen.
- Ist der Prozess nach einem `kill` immer noch nicht weg? Dann hat er sich geweigert zu sterben.  
→ In diesem Fall hilft die große Keule: `kill -9 16744`

### Vorsicht

Das ist die Ultima Ratio!

# Prozessverwaltung

killall – Prozesse mit bestimmtem Namen töten

```
$ killall firefox-bin
```

- Tötet *alle* Prozesse von `firefox`.
- Funktioniert sonst so wie `kill`.

## Vorsicht

Verhält sich auf nicht-Linux-System (z.B. Solaris) anders!

- 1 Prozessverwaltung
- 2 Pipes**
- 3 Rechteverwaltung
- 4 Secure Shell – Fernsteuern anderer Rechner
- 5 Kommandos, Kommandos, Kommandos, ...

# Pipes

## Ein- und Ausgabe-Streams

### Programme...

- lesen ihre Eingabe von der Standard-Eingabe (*stdin*)
- schreiben auf die Standard-Ausgabe (*stdout*)



# Pipes

## Ein- und Ausgabe-Streams

### Programme...

- lesen ihre Eingabe von der Standard-Eingabe (*stdin*)
- schreiben auf die Standard-Ausgabe (*stdout*)
- und schreiben Fehlermeldungen auf die Fehler-Ausgabe (*stderr*)



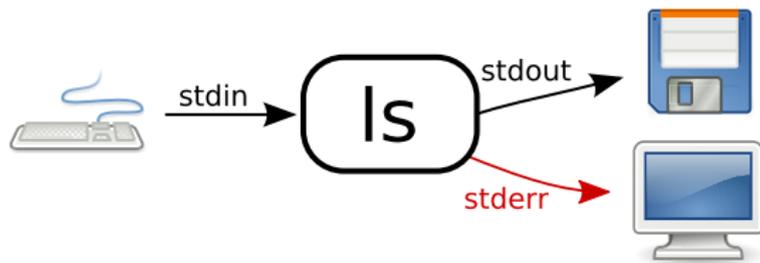
# Pipes

## Umleiten von Aus- und Eingabe-Streams



# Pipes

## Umleiten von Aus- und Eingabe-Streams



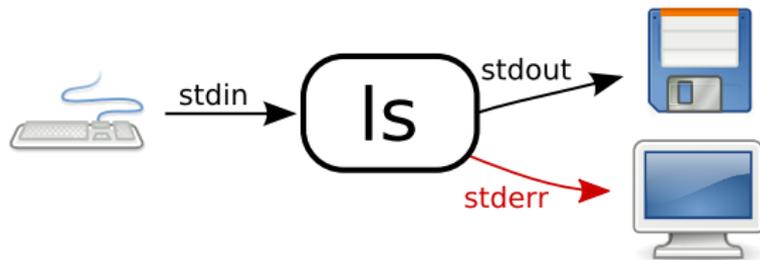
> – Ausgabe umleiten

> leitet *stdout* in eine Datei um.

⟨Befehl⟩ > ⟨Datei⟩

# Pipes

## Umleiten von Aus- und Eingabe-Streams



Beispiel: Erstellen einer Liste aller Dateien in einem Verzeichnis

```
$ ls
```

... und dann die Liste abtippen, oder:

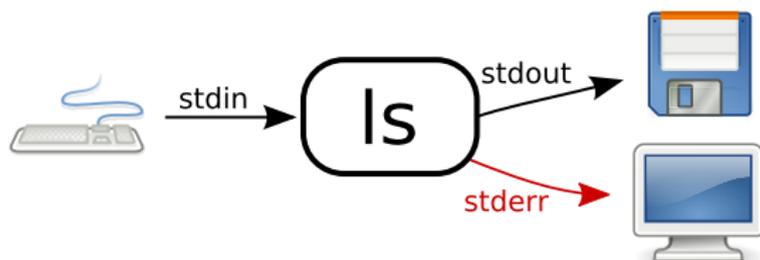
```
$ ls > listeMitDateien.txt
```

**Vorsicht**

> überschreibt den Inhalt einer Datei!

# Pipes

## Umleiten von Aus- und Eingabe-Streams



>> – Ausgabe umleiten (und anhängen)

>> leitet *stdout* in eine Datei um, dabei wird alles ans Ende der Datei angehängt.

Beispiel: Erstellen einer Liste aller Dateien aus zwei Verzeichnissen

```
$ ls bilder/ > listeMitDateien.txt  
$ ls urlaubsbilder/ >> listeMitDateien.txt
```

# Pipes

## Umleiten von Aus- und Eingabe-Streams



### < – Eingabe umleiten

< stellt den Inhalt einer Datei dem Programm auf *stdin* zur Verfügung.

`<Befehl> <Datei>`

# Pipes

## Umleiten von Aus- und Eingabe-Streams



Beispiel: Sortieren einer Liste von Dateien.

`sort` sortiert die Zeilen, die von `stdin` gelesen werden.

```
$ sort
```

... und dann die Liste der Dateien manuell eintippen oder:

# Pipes

## Umleiten von Aus- und Eingabe-Streams



Beispiel: Sortieren einer Liste von Dateien.

`sort` sortiert die Zeilen, die von `stdin` gelesen werden.

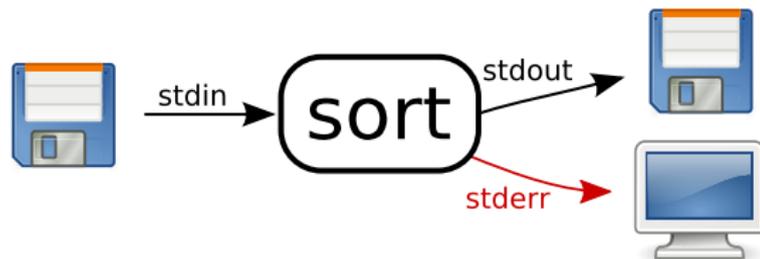
```
$ sort
```

... und dann die Liste der Dateien manuell eintippen oder:

```
$ sort < listeMitDateien.txt  
alex.jpg  
bruno.jpg  
...
```

# Pipes

## Umleiten von Aus- und Eingabe-Streams



```
$ sort < liste.txt > ausgabe.txt
```

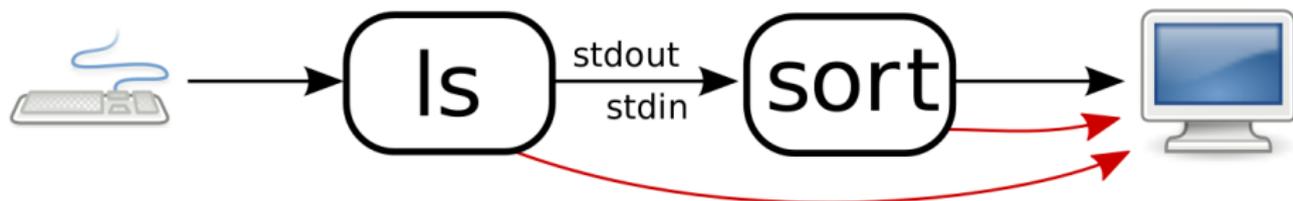
### Vorsicht

Falls Eingabe- und Ausgabedatei identisch sind, geschehen seltsame Dinge!

# Pipes

## Umleiten von Aus- und Eingabe-Streams

Natürlich kann man auch zwei Programme miteinander verbinden.



| – Ausgabe an ein anderes Programm weiterleiten

| („Pipe“) leitet *stdout* von einem Programm zum *stdin* eines anderen Programmes um.

`<Befehl 1> | <Befehl 2>`

# Pipes

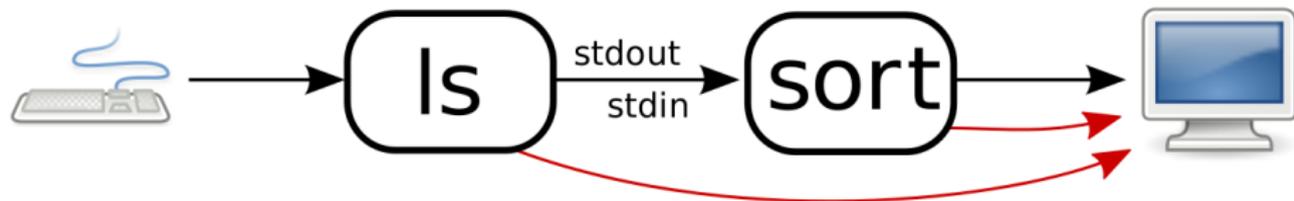
## Umleiten von Aus- und Eingabe-Streams

Umständlich: Sortieren einer Liste aller Dateien aus zwei Verzeichnissen.

```
$ ls bilder/ urlaubsbilder/ > listeMitDateien.txt  
$ sort < listeMitDateien.txt
```

# Pipes

## Umleiten von Aus- und Eingabe-Streams



Umständlich: Sortieren einer Liste aller Dateien aus zwei Verzeichnissen.

```
$ ls bilder/ urlaubsbilder/ > listeMitDateien.txt  
$ sort < listeMitDateien.txt
```

Besser: In einem Schritt mit Pipe:

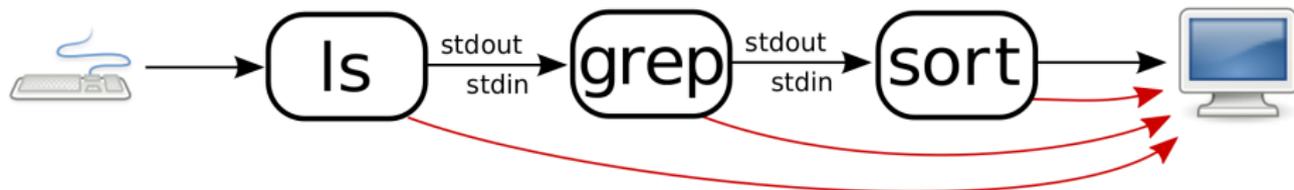
```
$ ls bilder/ urlaubsbilder/ | sort  
alex.jpg  
bruno.jpg
```

# Pipes

Umleiten von Aus- und Eingabe-Streams – beliebig erweiterbar!

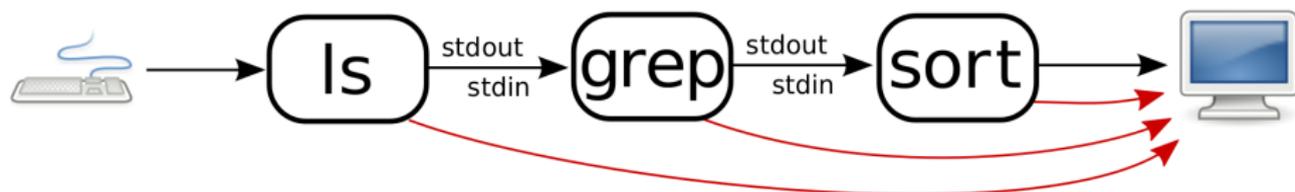
Beliebig erweiterbar!

- z. B. können wir eine sortierte Liste von Bildern nach allen Bildern von Peter durchsuchen!  
→ `grep` filtert die Liste



# Pipes

Umleiten von Aus- und Eingabe-Streams – beliebig erweiterbar!



Beispiel: Eine Liste von Bildern erstellen und diese durchsuchen

```
$ ls bilder/ | grep peter | sort  
peter.jpg  
peter_muende.jpg  
rainer-und-peter.jpg  
...
```

- 1 Prozessverwaltung
- 2 Pipes
- 3 Rechteverwaltung**
- 4 Secure Shell – Fernsteuern anderer Rechner
- 5 Kommandos, Kommandos, Kommandos, ...

Auf einem Mehrbenutzersystem wie dem Informatik-CIP mit **über 9000 Anwendern** sind sinnvolle Restriktionen essenziell:

- Man will seine *privaten Dokumente* vor fremden Augen schützen.
- Nur Administratoren sollen die *Konfiguration der Rechner* verändern können.
- *Kryptografie* setzt die bedingungslose Geheimhaltung der verwendeten Schlüssel voraus!

# Rechteverwaltung

Wofür brauchen wir Rechte?

Auf einem Mehrbenutzersystem wie dem Informatik-CIP mit **über 9000 Anwendern** sind sinnvolle Restriktionen essenziell:

- Man will seine *privaten Dokumente* vor fremden Augen schützen.
- Nur Administratoren sollen die *Konfiguration der Rechner* verändern können.
- *Kryptografie* setzt die bedingungslose Geheimhaltung der verwendeten Schlüssel voraus!

## Lösung: Der „Rechtsstaat“

- Für jede Datei und jedes Verzeichnis werden Berechtigungen vermerkt.
- Nur wer die entsprechenden Rechte besitzt, kann auf ein bestimmtes Objekt zugreifen.

## Gruppenkonzept

- Jeder Benutzer ist in mindestens einer Gruppe (im CIP: *immdstud*).
- Man kann in mehr als einer Gruppe sein.

### Gruppenkonzept

- Jeder Benutzer ist in mindestens einer Gruppe (im CIP: *immdstud*).
- Man kann in mehr als einer Gruppe sein.

### Einteilung der Benutzer

Pro Objekt im Dateisystem sind die Rechte für drei Klassen von Benutzern gespeichert:

**User** Diesem *Benutzer* „gehört“ die Datei / das Verzeichnis.  
Er darf Dateiberechtigungen vergeben.

**Group** Die Datei / das Verzeichnis ist dieser *Gruppe* zugeordnet.

**Others** Alle anderen.

### Und welche Berechtigungen hat eine Datei / ein Verzeichnis?

- `ls -l` zeigt eine ausführliche Ausgabe.
- Dabei zeigt die erste Spalte die Rechte an.
- Die dritte und vierte Spalte geben den Eigentümer bzw. die Eigentümergruppe an.

```
$ ls -l
-rw-r--r-- 1 simagrop immdstud 97 Oct 7 14:38 datei
-rwxr-x--- 1 simagrop immdstud 84 Oct 12 14:39 programm
```

# Rechteverwaltung

## Rechte auf Dateien

```
$ ls -l
-rw-r--r-- 1 simagrop imdstud 97 Oct 7 14:38 datei
-rwxr-x--- 1 simagrop imdstud 84 Oct 7 14:39 programm
```

### Und was heißt das jetzt?

- Das erste Zeichen zeigt den Typ an (z. B. d für ein Verzeichnis oder - für normale Dateien).
- Die nächsten drei Zeichen zeigen die Rechte für den **User**.
- Das zweite Zeichentripel zeigt die Rechte für die **Group**.
- Und (logischerweise) die verbleibenden drei Zeichen die Rechte für den Rest der Welt (**Others**).

# Rechteverwaltung

## Rechte auf Dateien

```
$ ls -l
-rw-r--r-- 1 simagrop imdstud 97 Oct 7 14:38 datei
-rwxr-x--- 1 simagrop imdstud 84 Oct 7 14:39 programm
```

r? w? x?

- r = lesbar
- w = schreibbar
- x = ausführbar

# Rechteverwaltung

## Rechte ändern

### Rechte ändern

- `chmod <mode> <Datei|Verzeichnis>`
- `chmod -R <mode> <Datei|Verzeichnis>` (rekursiv)

```
$ chmod      foo.bar
```

Das `mode`-Argument setzt sich zusammen aus drei Teilen

# Rechteverwaltung

## Rechte ändern

### Rechte ändern

- `chmod <mode> <Datei|Verzeichnis>`
- `chmod -R <mode> <Datei|Verzeichnis>` (rekursiv)

```
$ chmod g foo.bar
```

### Das `mode`-Argument setzt sich zusammen aus drei Teilen

#### Wen betrifft es?

u		Benutzer
g		Gruppe
o		Rest
a		alle

# Rechteverwaltung

## Rechte ändern

### Rechte ändern

- `chmod <mode> <Datei|Verzeichnis>`
- `chmod -R <mode> <Datei|Verzeichnis>` (rekursiv)

```
$ chmod g+ foo.bar
```

### Das `mode`-Argument setzt sich zusammen aus drei Teilen

#### Wen betrifft es?

u	Benutzer
g	Gruppe
o	Rest
a	alle

#### Welche Aktion?

+	Rechte geben
-	Rechte wegnehmen
=	Rechte setzen

# Rechteverwaltung

## Rechte ändern

### Rechte ändern

- `chmod <mode> <Datei|Verzeichnis>`
- `chmod -R <mode> <Datei|Verzeichnis>` (rekursiv)

```
$ chmod g+rx foo.bar
```

### Das `mode`-Argument setzt sich zusammen aus drei Teilen

#### Wen betrifft es?

u	Benutzer
g	Gruppe
o	Rest
a	alle

#### Welche Aktion?

+	Rechte geben
-	Rechte wegnehmen
=	Rechte setzen

#### Welche Rechte?

r	lesen
w	schreiben
x	ausführen

# Rechteverwaltung

## Änderungen im Rechtssystem

```
$ chmod u+r datei
```

```
$ chmod go-rwx datei
```

```
$ chmod a+rx datei
```

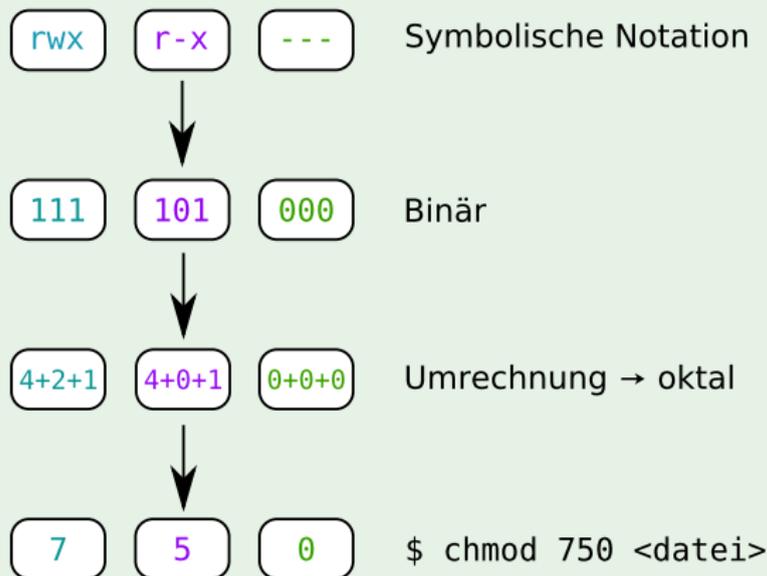
```
$ chmod u=rwx,g=rx,o= datei
```

# Rechteverwaltung

## Änderungen im Rechtssystem

Aber ich kann besser mit Zahlen als mit Zeichen!

Kein Problem:



# Rechteverwaltung

## Rechte auf Verzeichnissen

```
$ ls -l
-rw-r--r-- 1 simagrop immdstud 97 Oct 7 14:38 datei
drwxr-xr-x 2 simagrop immdstud 40 Oct 7 14:37 verzeichnis
```

### Aber Moment! Wie können Verzeichnisse ausführbar sein?

Für Verzeichnisse gilt:

- Lesbar: Inhalt des Verzeichnisses kann aufgelistet werden. (z. B. mit `ls`)
- Schreibbar: Weitere Verzeichnisse und Dateien können angelegt bzw. *gelöscht* werden.
- Ausführbar: Verzeichnis kann betreten werden ( $\Rightarrow$  Kein Zugriff auf darin befindliche Dateien und Verzeichnisse).

# Rechteverwaltung

## Rechte auf Verzeichnissen

```
$ ls -l verzeichnis/  
-rw- --- --- 1 simagrop imdstud 97 Oct 7 14:38 dokument  
d rwx --- --- 1 simagrop imdstud 97 Oct 7 14:38 ordner  
$ chmod -R a+rx verzeichnis/
```

Was passiert jetzt in verzeichnis?

# Rechteverwaltung

## Rechte auf Verzeichnissen

```
$ ls -l verzeichnis/  
-rw- --- --- 1 simagrop imdstud 97 Oct 7 14:38 dokument  
drwx --- --- 1 simagrop imdstud 97 Oct 7 14:38 ordner  
  
$ chmod -R a+rx verzeichnis/
```

Was passiert jetzt in verzeichnis?

```
$ ls -l verzeichnis/  
-rwxr-xr-x 1 simagrop imdstud 97 Oct 7 14:38 dokument  
drwxr-xr-x 1 simagrop imdstud 97 Oct 7 14:38 ordner
```

Ups...

# Rechteverwaltung

## Rechte auf Verzeichnissen

Stattdessen:

```
$ ls -l verzeichnis/  
-rw- --- --- 1 simagrop immdstud 97 Oct 7 14:38 dokument  
d rwx --- --- 1 simagrop immdstud 97 Oct 7 14:38 ordner  
  
$ chmod -R a+rX verzeichnis/  
  
$ ls -l verzeichnis/  
-rw-r--r-- 1 simagrop immdstud 97 Oct 7 14:38 dokument  
d rwxr-xr-x 1 simagrop immdstud 97 Oct 7 14:38 ordner
```

`chmod -R +X`

- Setzt das x-Recht nur dort, wo schon für irgendeinen Benutzer x-Rechte eingetragen sind.
- Also normalerweise nur bei Verzeichnissen und Programmdateien.

In der Standardeinstellung steht das Home-Verzeichnis jedes Benutzers sperrangelweit offen!

```
$ ls -ld ~  
drwxr-xr-x 179 simagrop immdstud 17408 2010-10-04 18:31 /home/cip/2005/simagrop
```

- Andere Benutzer können den Inhalt eures Home-Verzeichnisses zwar nicht verändern, aber immerhin durchsuchen.
- Nur wichtige Konfigurationsdateien sind automatisch geschützt.
- **Ihr seid für die Sicherheit eurer Daten selber verantwortlich!**

In der Standardeinstellung steht das Home-Verzeichnis jedes Benutzers sperrangelweit offen!

```
$ ls -ld ~  
drwxr-xr-x 179 simagrop imdstud 17408 2010-10-04 18:31 /home/cip/2005/simagrop
```

- Andere Benutzer können den Inhalt eures Home-Verzeichnisses zwar nicht verändern, aber immerhin durchsuchen.
- Nur wichtige Konfigurationsdateien sind automatisch geschützt.
- **Ihr seid für die Sicherheit eurer Daten selber verantwortlich!**

**Auf Nummer sicher gehen:**

```
$ chmod 700 ~
```

# Rechteverwaltung

## Änderungen im Rechtssystem

### Eigentümer ändern

- `chown <login> <Datei|Verzeichnis>`
- `chown -R <login> <Datei|Verzeichnis>` (rekursiv)

Das darf aber nur root!



# Rechteverwaltung

## Änderungen im Rechtssystem

### Eigentümer ändern

- `chown <login> <Datei|Verzeichnis>`
- `chown -R <login> <Datei|Verzeichnis>` (rekursiv)

Das darf aber nur root!



### Eigentümergruppe ändern

- `chgrp <group> <Datei|Verzeichnis>`
- `chgrp -R <group> <Datei|Verzeichnis>` (rekursiv)

# Rechteverwaltung

## Access Control Lists (ACLs)

Aber geht's nicht etwas feingranularer?

Mein AuD-Verzeichnis soll nur mein Übungspartner lesen können!

```
$ ls -l
drwxr-xr-x+ 1 simagrop imdstud 0 2009-10-11 10:45 aclverzeichnis

$ setfacl -R -m u:sijesche:rX aclverzeichnis
$ setfacl -R -d -m u:sijesche:rX aclverzeichnis

$ getfacl aclverzeichnis
# file: aclverzeichnis
# owner: simagrop
# group: imdstud
user::rwx          default:user::rwx          default:mask::r-x
user:sijesche:r-x  default:user:sijesche:r-x
group::r-x         default:group::r-x
other::r-x         default:other::r-x
```

# Rechteverwaltung

## Access Control Lists (ACLs)

Aber geht's nicht etwas feingranularer?

Mein AuD-Verzeichnis soll nur mein Übungspartner lesen können!

```
$ ls -l  
drwxr-xr-x+ 1 simagrop imdstud 0 2009-10-11 10:45 aclverzeichnis
```

```
$ setfacl
```

```
$ setfacl
```

```
$ getfacl
```

```
# file
```

```
# owner
```

```
# group
```

```
user::rwx      default:user::rwx      default:mask::r-x
```

```
user:sijesche:r-x  default:user:sijesche:r-x
```

```
group::r-x      default:group::r-x
```

```
other::r-x      default:other::r-x
```

\$ man getfacl  
oder *Nautilus* verwenden

- 1 Prozessverwaltung
- 2 Pipes
- 3 Rechteverwaltung
- 4 Secure Shell – Fernsteuern anderer Rechner**
- 5 Kommandos, Kommandos, Kommandos, ...

# Secure Shell – Fernsteuern anderer Rechner

Remote-Shell mit verschlüsselter Verbindung

- SSH ermöglicht das Fernsteuern eines anderen Rechners über ein virtuelles Terminal.
- Alle Befehle, die man in dieses Terminal eintippt, werden auf dem Remote-Rechner ausgeführt.

# Secure Shell – Fernsteuern anderer Rechner

## Remote-Shell mit verschlüsselter Verbindung

- SSH ermöglicht das Fernsteuern eines anderen Rechners über ein virtuelles Terminal.
- Alle Befehle, die man in dieses Terminal eintippt, werden auf dem Remote-Rechner ausgeführt.

### Verbinden mit einem CIP-Rechner

```
ssh <login>@<rechnername>.cs.fau.de
```

- <login> ist dein CIP-Login.
- <rechnername> ist beispielsweise faui06c (findet man auf den Monitoren).
- Benutze dein CIP-Passwort, um dich einzuloggen.

### Tipp

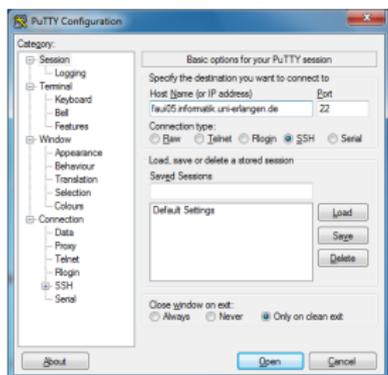
Im CIP ohne <login>@ und .cs.fau.de möglich z. B.: `ssh faui06c`

# Secure Shell – Fernsteuern anderer Rechner

## SSH unter Windows

### Unter Windows

- Für Windows existieren verschiedene SSH-Programme.
- Wir empfehlen PuTTY: <http://www.putty.org/>
- Auch hier <rechnername>.cs.fau.de als Host
- Einloggen mit CIP-Account



```
fau05.informatik.uni-erlangen.de - PuTTY
Overfull \vbox (4.2427pt too high) detected at line 271
[17] [18] [19] (./folien-v1-tag2.vrb) [20] [21] [22] [23] [24]
(/usr/share/texmf/tex/latex/R/amsctm.fd) [25] [26] (./folien-v1-tag2.vrb)
[27] [28] <images/opensh.png, id=245, 580.1675pt x 191.71625pt>
<use images/opensh.png> [29 <./images/opensh.png>] [30] [31] [32 <./images/pu
tty.png>] [33] [34] [35 <./images/vinscp.png>] (./folien-v1-tag2.aux)

LaTeX Font Warning: Size substitutions with differences
(Font)
up to 1.0pt have occurred.

)
(see the transcript file for additional information)</usr/share/texmf-tetex/fo
nts/typel/bluesky/cm/cmaltt10.pfb></usr/share/texmf-tetex/fonts/typel/bluesky/cm
/cmaltt10.pfb></usr/share/texmf-tetex/fonts/typel/bluesky/cm/cmss12.pfb></usr/s
hare/texmf-tetex/fonts/typel/bluesky/symbols/msam10.pfb></usr/share/texmf-tetex
/fonts/typel/bluesky/cm/cmss10.pfb></usr/share/texmf-tetex/fonts/typel/bluesky/
cm/cmss10.pfb></usr/share/texmf-tetex/fonts/typel/bluesky/cm/cmalt1.pfb></usr
/share/texmf-tetex/fonts/typel/bluesky/cm/cmalt10.pfb></usr/share/texmf-tetex/fo
nts/typel/bluesky/cm/cmss110.pfb></usr/share/texmf-tetex/fonts/typel/bluesky/cm
/cmss8.pfb></usr/share/texmf-tetex/fonts/typel/bluesky/cm/cmss10.pfb></usr/shar
e/texmf-tetex/fonts/typel/bluesky/cm/cmss12.pfb>
Output written on folien-v1-tag2.pdf (35 pages, 374692 bytes).
Transcript written on folien-v1-tag2.log.
fau05 [~/proj/cipmp/simagrop/fsi/vorkurs/0809wa/tag2/vorlesung]>
```

# Secure Shell – Fernsteuern anderer Rechner

## Kopieren von Dateien zwischen Rechnern

### *scp* (*Secure copy*)

scp kann Dateien von einem Rechner auf einen anderen kopieren und verwendet ssh für die Authentifizierung.

### Unter Linux

```
scp <quelle> <ziel>
```

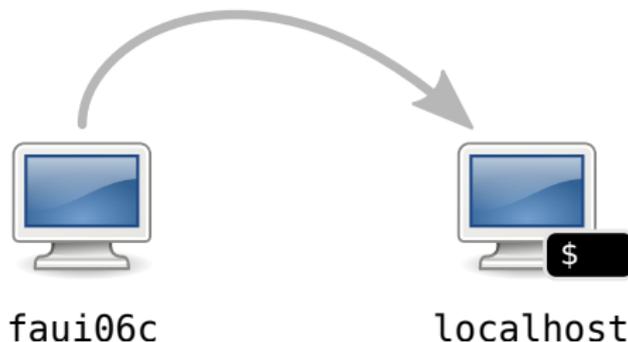
```
⇒ <quelle>/<ziel>: <login>@<rechnername>:<pfad+name>
```

### Wichtig!

- Keine Leerzeichen in <quelle> oder <ziel>!
- Alles ohne Doppelpunkt wird als *lokale* Datei interpretiert!

# Secure Shell – Fernsteuern anderer Rechner

## Kopieren von Dateien zwischen Rechnern – Beispiele

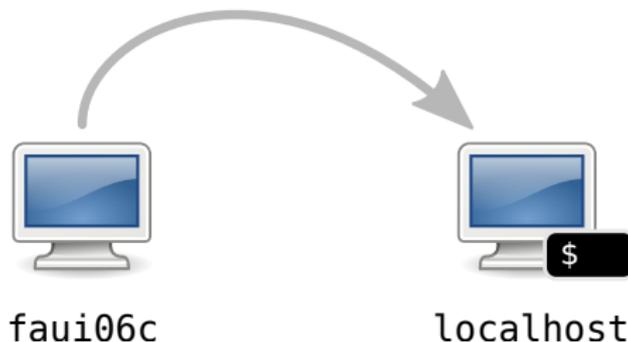


```
$ scp simagrop@faui06c:datei .
```

Kopiert die Datei `datei` aus dem *Home* von `simagrop` in das aktuelle Verzeichnis (durch den `'.'` angegeben).

# Secure Shell – Fernsteuern anderer Rechner

## Kopieren zwischen Rechnern – Beispiele

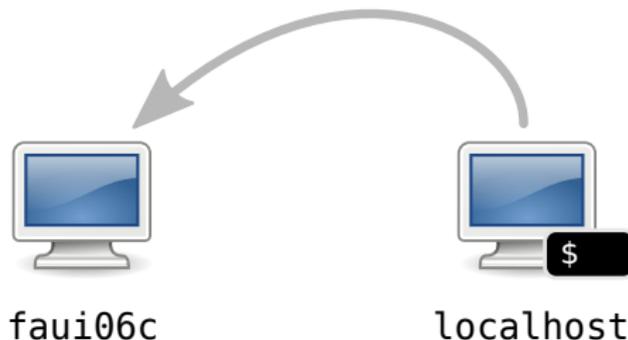


```
$ scp simagrop@faui06c:/tmp/foo.bar .
```

Kopiert die Datei /tmp/foo.bar vom Rechner faui06c ins aktuelle Verzeichnis.

# Secure Shell – Fernsteuern anderer Rechner

## Kopieren zwischen Rechnern – Beispiele



```
$ scp foo.bar simagrop@faui06c:/tmp
```

Kopiert die Datei `foo.bar` ins Verzeichnis `/tmp` auf dem Rechner `faui06c`.

# Secure Shell – Fernsteuern anderer Rechner

## Kopieren zwischen Rechnern – Beispiele



```
$ scp faui06a:/tmp/foo.bar faui06b:/tmp
```

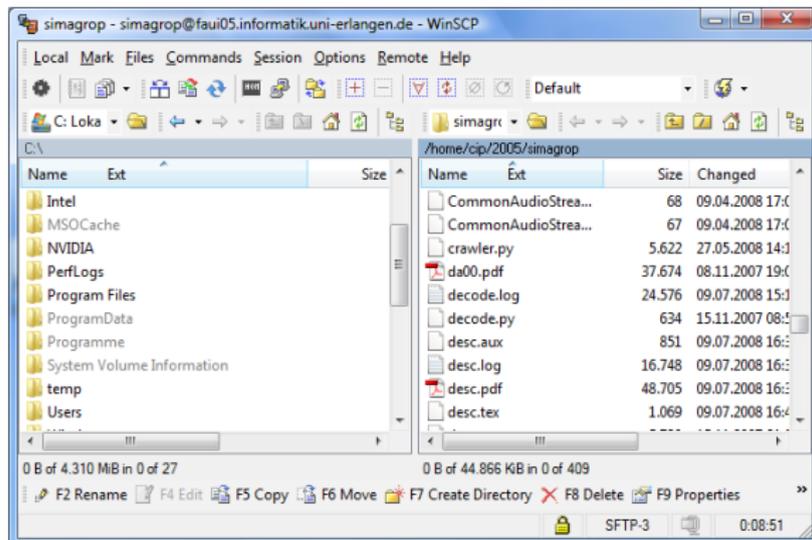
Kopiert die Datei `/tmp/foo.bar` von der `faui06a` ins gleiche Verzeichnis auf der `faui06b` (mit dem aktuellen Benutzer).

# Secure Shell – Fernsteuern anderer Rechner

Und wenn ich Dateien mit meiner heimischen Windows-Kiste austauschen will?

## Unter Windows

Um Dateien zwischen Linux und Windows auszutauschen, kann man WinSCP verwenden (<http://winscp.net/>).



# Secure Shell – Fernsteuern anderer Rechner

Vereinfachungen für den Alltag mit SSH

```
$ ssh simagrop@fau06c.cs.fau.de
```

- Meist muss man sich verschiedene Kombinationen von Benutzernamen und Rechnern merken.

# Secure Shell – Fernsteuern anderer Rechner

Vereinfachungen für den Alltag mit SSH

```
$ ssh simagrop@fau06c.cs.fau.de
```

- Meist muss man sich verschiedene Kombinationen von Benutzername und Rechner merken.
- Lösung: Trage diese Informationen in die Konfigurationsdatei ein.
- Wo ist die Datei: `~/.ssh/config`

# Secure Shell – Fernsteuern anderer Rechner

Vereinfachungen für den Alltag mit SSH

```
$ ssh simagrop@faiu06c.cs.fau.de
```

- Meist muss man sich verschiedene Kombinationen von Benutzername und Rechner merken.
- Lösung: Trage diese Informationen in die Konfigurationsdatei ein.
- Wo ist die Datei: `~/.ssh/config`

## Beispiel: Definiere eine Verbindung *cip*

```
Host cip
  HostName faiu06c.cs.fau.de
  User simagrop
```

```
$ ssh cip
```

- Hostnamen der CIP-Rechner, welche 24/7 verfügbar sind:  
`faiu06a–h`, `faiu00a–t`

# Kommandos, Kommandos, Kommandos, ...

- 1 Prozessverwaltung
- 2 Pipes
- 3 Rechteverwaltung
- 4 Secure Shell – Fernsteuern anderer Rechner
- 5 Kommandos, Kommandos, Kommandos, ...**

# Kommandos, Kommandos, Kommandos, ...

## Befehlsübersicht

Unter Unix gibt es noch viele viele weitere nützliche Kommandos. Die hier vorgestellte Auswahl maßt sich nicht an, auch nur annähernd vollständig zu sein!

## Befehlsübersicht

<code>find</code>	suchen nach Dateien
<code>grep</code>	suchen in Dateien
<code>tar/unzip/unp</code>	Packtools
<code>sort</code>	sortieren
<code>uniq</code>	Zeilen zusammenfassen
<code>head/tail</code>	erste bzw. letzte Zeilen ausgeben
<code>cut</code>	Ausschneiden von Spalten
<code>awk</code>	Programmiersprache zum Bearbeiten von Textdaten
<code>wc</code>	Wörter zählen

# Kommandos, Kommandos, Kommandos, ...

find – Suche nach Dateien

Rekursive Dateisuche in Verzeichnisstrukturen nach bestimmten Kriterien.

## Aufruf

```
$ find [dir] <filter1> <filter2> ...
```

Häufig benutzte Filter:

- name, -iname sucht mit Wildcards nach Dateinamen, mit *i* case-insensitive (Groß-/Kleinschreibung egal)
- type [f|d] sucht nur nach bestimmten Dateityp, *f* für Files, *d* für Directories

# Kommandos, Kommandos, Kommandos, ...

find – Suche nach Dateien

```
$ find . -name '*.pdf'  
./AuD/uebungen/blatt01.pdf  
./studbesch_ws1314.pdf  
  
$ find Musik/ -iname '*.mp3'  
./Musik/Deep_Purple/Made_in_Japan/Highwaystar.MP3
```

# Kommandos, Kommandos, Kommandos, ...

## grep – Suchen in Dateien

Sucht in der Standard-Eingabe (*stdin*) oder in Dateien nach Zeilen, die auf einen regulären Ausdruck passen, und gibt passende Zeilen auf der Standard-Ausgabe (*stdout*) aus.

Der einfachste reguläre Ausdruck umfasst nur das Suchwort selbst.

### Aufruf

```
grep <pattern> [file1 file2 ...]
```

```
$ grep ssh /etc/services
ssh      22/tcp      # SSH Remote Login Protocol
ssh      22/udp
```

### Tipp

```
grep -i <pattern>
```

Option *-i* zum Suchen ohne Beachtung der Groß-/Kleinschreibung.

# Kommandos, Kommandos, Kommandos, ...

tar – Packtools

Mehrere Dateien packen und entpacken (ähnlich zu .zip-Dateien).  
Erstellte Dateien heißen *Tarballs*.

## Aufruf

```
tar [optionen] <pfade> ...
```

Häufig benutzte Optionen:

- c** Create, erstellt neues Archiv
- x** Extract, entpackt bestehendes Archiv
- v** Verbose, zeigt welche Dateien behandelt werden
- f** Filename, gibt an wie das Tarball heißt
- z** Komprimierung mit gzip
- j** Komprimierung mit bzip2

# Kommandos, Kommandos, Kommandos, ...

Packtools – Beispiele

Zu einem *Tarball* packen:

```
$ tar -czf foo.tar.gz foo/
```

Einen *Tarball* entpacken:

```
$ tar -xzf foo.tar.gz
```

zip-Dateien entpacken:

```
$ unzip file.zip
```

... und ein Shellsript, das so gut wie alle Archive entpacken kann:

```
$ unp file
```

# Kommandos, Kommandos, Kommandos, ...

sort – Sortieren

Sortiert die Standard-Eingabe (*stdin*) und gibt die sortierte Liste auf der Standard-Ausgabe (*stdout*) aus.

## Aufruf

```
sort [options] < infile  
<andererbefehl> | sort [options]
```

Häufig benutzte Optionen:

- r Sortiert in umgekehrter Reihenfolge
- n Sortiert numerisch statt alphabetisch

# Kommandos, Kommandos, Kommandos, ...

uniq – Nur Unikate

uniq fasst aufeinander folgende gleiche Zeilen zusammen.

input.txt

```
foo
bar
bar
baz
foo
```

```
$ uniq < input.txt
foo
bar
baz
foo
```

```
$ sort < input.txt | uniq
bar
baz
foo
```

# Kommandos, Kommandos, Kommandos, ...

`head` und `tail` – Nur den Anfang bzw. das Ende, bitte

Gibt nur die ersten bzw. letzten  $N$  Zeilen der Standard-Eingabe (*stdin*) auf der Standard-Ausgabe (*stdout*) aus.

```
$ head -n3 < /etc/services
# Network services, Internet style
#
# Note that it is presently the policy of IANA ...
```

```
$ tail -n3 < /etc/services
# Local services
submission      587/tcp        # Mail Message Submission
submission      587/udp        # see RFC 2476
```

# Kommandos, Kommandos, Kommandos, ...

cut – Spalten ausschneiden

Filtert bestimmte Spalten aus der Standard-Eingabe (*stdin*) und gibt sie auf der Standard-Ausgabe (*stdout*) aus.

## Aufruf

```
cut [-d <delimiter>] -f <fields>
```

**delimiter** ist ein einzelnes Zeichen das zur Trennung der Spalten benutzt wird. Standardwert ist TAB.

**fields** bezeichnet welche Spalten ausgegeben werden sollen, separiert durch Komma (1,2,3) oder Bereiche (1-3).

# Kommandos, Kommandos, Kommandos, ...

cut – Spalten ausschneiden

```
$ head -n3 < /etc/passwd  
root:x:0:0:root:/root:/bin/bash  
daemon:x:1:1:daemon:/usr/sbin:/bin/sh  
bin:x:2:2:bin:/bin:/bin/sh
```

```
$ cut -d: -f 1,5 < /etc/passwd | head -n3  
root:root  
daemon:daemon  
bin:bin
```

# Kommandos, Kommandos, Kommandos, ...

awk – supercool & supertoll

Mächtige Programmiersprache, praktisch für Filteroperationen. Der Umfang könnte aber eine eigene Vorlesung füllen.

## Aufruf

```
awk <script>
```

# Kommandos, Kommandos, Kommandos, ...

## Praktische Anwendung

Top 10 der Shell-History:

```
$ history |awk '{print $2}' |sort |uniq -c |sort -rn |head
1201 emacs
 886 make
 881 ls
 848 cd
 796 git
 643 svn
 166 cat
 159 rm
 150 man
```

# Kommandos, Kommandos, Kommandos, ...

`wc` – Wörter zählen

`wc` zählt in der Standard-Eingabe (*stdin*) wahlweise Anzahl der Zeilen, Wörter oder Zeichen und gibt die Anzahl auf der Standard-Ausgabe (*stdout*) aus.

Die Ausgabe von `wc` umfasst 3 Spalten (Zeilen, Wörter und Zeichen):

```
$ wc /etc/services
559 2536 18414 /etc/services
```

Mit `wc -l` werden nur Zeilen gezählt:

```
$ ls | wc -l
72
```

Fragen?

42

## Referenzen

- [http://fsi.informatik.uni-erlangen.de/repo\\_public/vorkurs/](http://fsi.informatik.uni-erlangen.de/repo_public/vorkurs/)
- Aufzeichnung zu finden unter: <http://video.cs.fau.de/>