Vorkurs Linux-Grundlagen

Vorkurs Linux / Informatik
Tag 4

Heute:

- Von zu Hause mit dem Uni-Rechner verbinden
- Voreinstellungen für die Kommandozeile
- Kommandozeilen-Programmierung
  - „kleine Programme“: Aliase
  - if-Abfragen und for-Schleifen
  - Kommandozeilen-Skripte
  - Live-Program: Script für nutzlose Image-Macros :)

Wiederholung: Prozesse

ablaufende Programme = Prozesse

Typische Operationen:

- laufende Prozesse anzeigen
- (unerwünschte) Prozesse beenden

Werkzeuge: ps, top, kill, nice

Wiederholung: Dateiverwaltung

Dateien verwalten

- vieles kennen wir schon: pwd, ls, cd, cp, mv, rm, ...

Weitere typische Werkzeuge

- Lese-/Schreibrechte verwalten (id, chmod)
- Dateien nach Namen finden (find)
- Komprimieren und archivieren (gzip, bzip2, xz, tar)

Wiederholung: E-Mail

- Netiquette
- Schwächen von SMTP
- Kryptographische Unterschriften

Was machen wir heute:

- Von zu Hause mit dem Uni-Rechner verbinden
- Voreinstellungen für die Kommandozeile
- Kommandozeilen-Programmierung
  - „kleine Programme“: Aliase
  - if-Abfragen und for-Schleifen
  - Kommandozeilen-Skripte
  - Live-Program: Script für nutzlose Image-Macros :)

Block 4: Einloggen

Typische Aufgaben

- Wie komme ich von zu Hause auf die Uni-Rechner?
- Wer sitzt gerade noch am Rechner?
- Datenaustausch von zu Hause mit dem Uni-Rechner

Von zu Hause am Uni-Rechner arbeiten

ssh : secure shell

```
> ssh porta.techfak.uni-bielefeld.de -l <name>
```

der einzige von außen zugängliche Uni-Rechner

euer Benutzername

porta hat nicht genug Rechenleistung
→ von dort mit ssh auf andere Rechner weiterverbinden

siehe auch:
http://www.techfak.uni-bielefeld.de/ags/rbg/de/rechner-unix-porta.html

X-Sessions über ssh tunneln

Voraussetzung: Euer Client hat X11

- Linux oder *BSD: ✔
- Mac OS X: optional ab 10.5 dabei
- Windows: geht nicht ✘

- außerdem: schnelle Netzverbindung

Aufruf:

```
> ssh porta.techfak.uni-bielefeld.de -X -l <name>
```

Auf Linux-Rechner wechseln

ssh : secure shell

```
> ssh leonardo
```

wechselt auf anderen Linux-Rechner (hier: leonardo), um dort Programme auszuführen

Es gibt eine Sammeladresse für den Server-Pool:

```
> ssh linux.compute.techfak.uni-bielefeld.de
> ssh linux.compute
```

Lokalen Rechner untersuchen

who : show who is logged on

```
> who
```

zeigt alle angemeldeten („logged in“) Benutzer

- nur auf dem lokalen Rechner
- in großen Netzwerken nur begrenzt hilfreich

Datenaustausch mit zu Hause (1)

scp : secure copy

von zu Hause auf den Uni-Rechner übertragen

euer Benutzername („login“)

```
> scp datei <Benutzername>@porta.techfak.uni-bielefeld.de:
```

die zu übertragende Datei

(der einzige) von außen zugängliche Uni-Rechner

Datenaustausch mit zu Hause (2)

vom Uni-Rechner Daten nach Hause holen:

```
> scp <benutzername>@porta.techfak.uni-bielefeld.de:datei .
```

Möglichkeiten unter Windows

WinSCP: nur Dateien übertragen

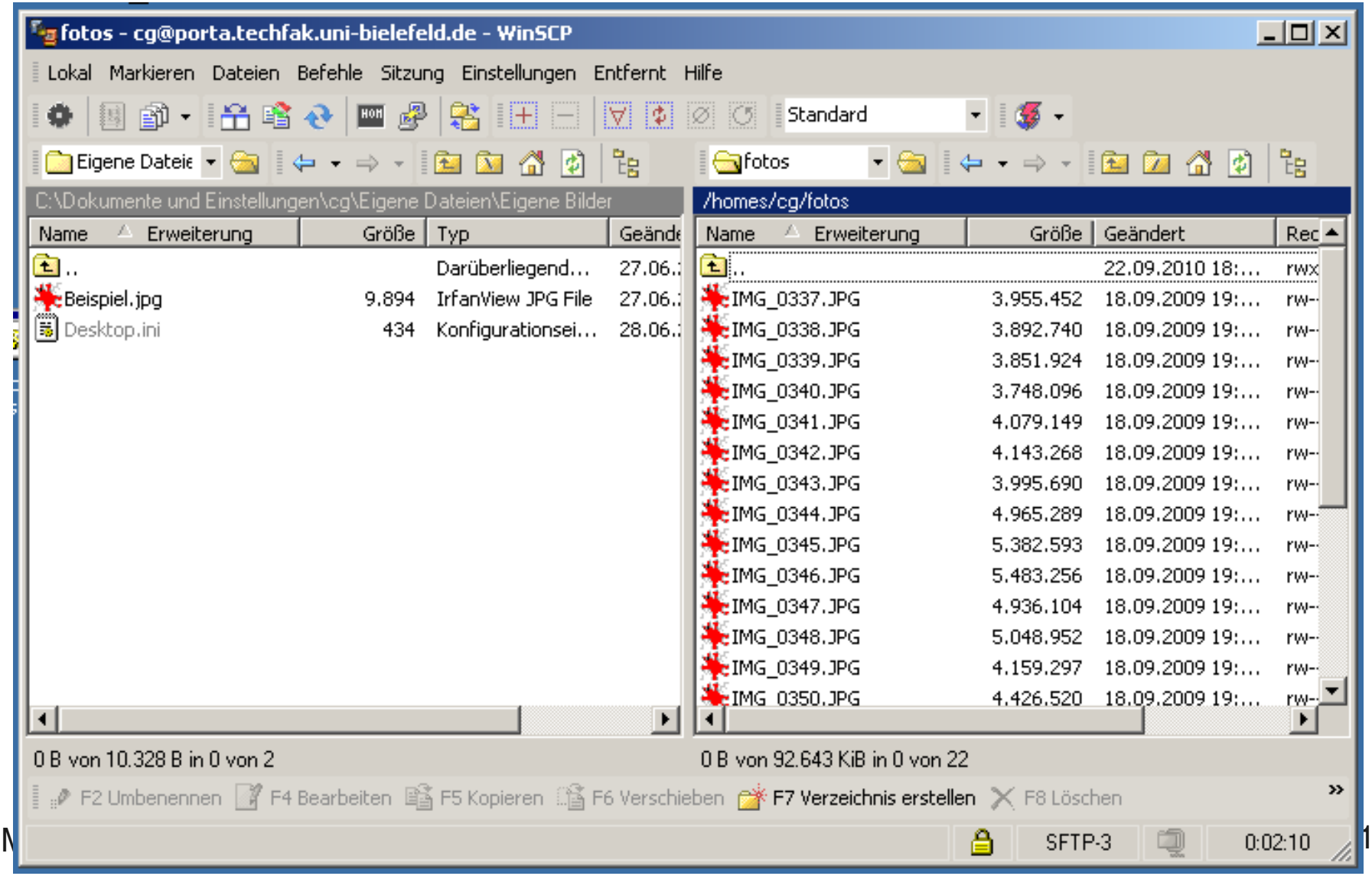
http://winscp.net/de

PuTTY: Terminal und Dateien übertragen

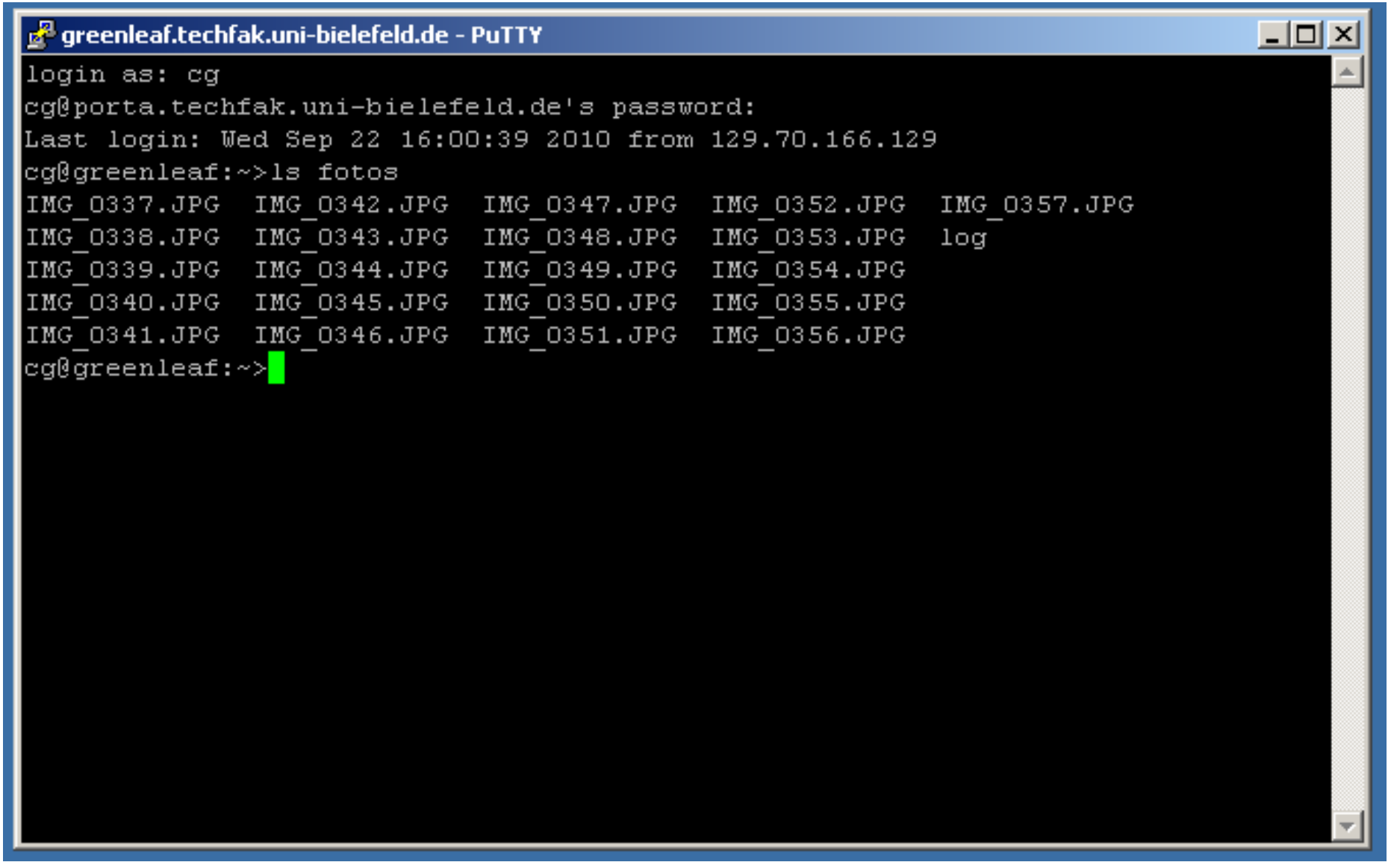
http://www.chiark.greenend.org.uk/~sgtatham/putty/

nicht möglich: Fensterweiterleitung

Beispiel: WinSCP



Beispiel: PuTTY



Block 5: Shell-Programmierung

Typische Aufgaben

- Voreinstellungen für häufig verwendete Programme
- Abfragen und Schleifen programmieren
- Kommandozeilen-Skripte

→ am Beispiel der Bourne-Shell (bash)

Aliase (1)

alias : Ein Programm unter einem anderen Namen verwenden

Motivation: Immer ls -l tippen ist lästig

```
> alias ll="ls -l"
> ll
```

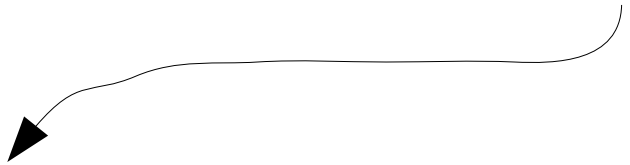
`ls -l` wird als neuer Befehl mit dem Namen `ll` definiert

→ in Zukunft kann man einfach `ll` tippen

Aliase (2)

Genauere Betrachtung:

```
> alias s2="sort -k 2 -n"
```



```
> s2 planeten.txt
```

→ Shell ersetzt linke Seite der Gleichung (s2)

durch rechte Seite (sort -k 2 -n)

→ alias kann weitere Aufrufwerte haben (planeten.txt)

Aliase automatisch setzen

.bashrc : „bash resources“ - Konfigurationsdatei

- wird beim Starten der Kommandozeile ausgeführt

→ was ihr in die .bashrc hineinschreibt gilt

als hättet ihr es direkt eingegeben

„Aktivieren“ der Änderungen:

- gelten in jedem Fenster, das ihr danach öffnet
- **`> source .bashrc`** (im Benutzerverzeichnis)

Warnung

Vermurkste .bashrc

→ kein vernünftiges Arbeiten mehr möglich

Vorsichtsmaßnahmen

- **`> cp .bashrc .bashrc-alt`** , dann ändern!
- besser: Versionskontrolle
- Änderungen durch öffnen eines neuen Fensters prüfen!
- Editor erst schließen, wenn Änderungen okay sind!

Shell-Skripte (1)

Shell-Skript = Datei, die Kommandozeilen-Programme aufruft

Prinzipieller Aufbau der Datei:

```
# ! /bin/bash
```

Shell zum Ausführen des Skriptes

```
echo Hallo
echo hier
echo bin ich
```

Aufrufe, wie ihr sie auch direkt tippen würdet

Shell-Skripte (2)

Ausführen von Shell-Skripten

1. Möglichkeit:

```
> source skript.bash
```

2. Möglichkeit:

```
> chmod u+x skript.bash
> ./skript.bash
```

Suchpfade einstellen (1)

Shell-Skripte verhalten sich wie „echte“ Programme

→ warum dann > `./skript.bash`

und nicht einfach `> skript.bash ?`

Die Shell hat folgende „Quellen“ für Programme:

- eingestellte „Suchpfade“
- Programme aus direkt angegebenen Pfaden (./)

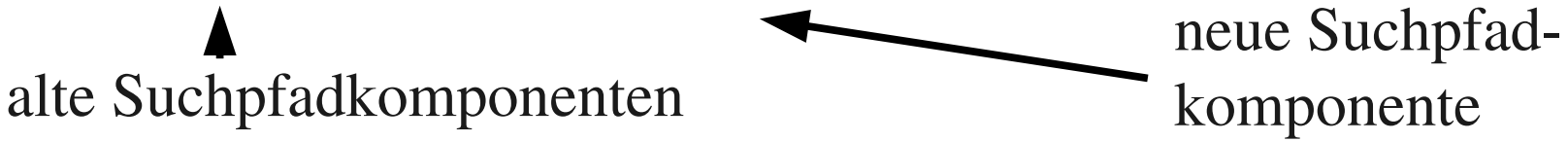
Suchpfade einstellen (2)

Idee: spezielles Verzeichnis für Skripte einrichten

```
> mkdir shell-skripte
> mv skript.bash shell-skripte
```

... und dann in den Suchpfad der Shell aufnehmen:

```
> PATH=$PATH:~/shell-skripte
```



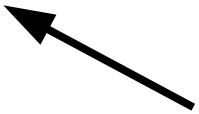

```
> skript.bash
```
(und ausprobieren)

Suchpfade einstellen (3)

Vorsicht: vermurkster Suchpfad → alle Programme „weg“

(Programme sind noch da, aber die Shell findet sie nicht mehr)

```
PATH=$PATH:~/shell-skripte
```



nicht vergessen (beliebte Falle ;-)

Erste Hilfe: absolute Pfade benutzen, z.B.

```
> /bin/ls
> /usr/bin/emacs
```

Suchpfade einstellen (4)

Ein dicker Bock, den man niemals schießen darf:

- ~~den Punkt . in den Suchpfad aufnehmen~~

Im Verzeichnis /tmp gebe es folgendes Skript

```
#!/bin/bash
rm -rf ~/*                    #löscht das Benutzerverz.
```

und zwar mit dem Namen "**ls**".

Würdet ihr dort **./ls** aufrufen? Nein?
Dann nehmt . nicht in euren Suchpfad auf!

Argumente an Shell-Skripte übergeben

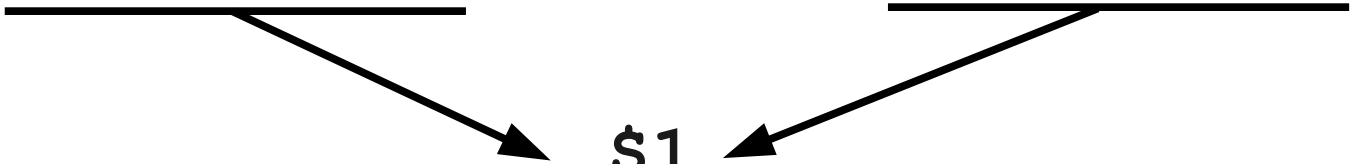
Beispiel zur Übergabe von Argumenten an Shell-Skripte:

```
#!/bin/bash

echo "Erstes  : $1"
echo "Zweites: $2"
echo "Drittes: $3"

echo "Anzahl:  $#"
echo "Alle:    $*"
```

Beispiel für Parameterübergabe

Zur Erinnerung aus Tag 2:

```
head -2 planeten2.txt; tail -n +3 planeten2.txt | sort
```



Abstrahieren und zusammenfassen

```
#!/bin/bash
head -2 $1 ; tail -n +3 $1 | sort

> hsort.bash planeten2.txt
```

Bedingte Ausführung (1)

Bedingte Ausführung: if ... then ... else
oder auf Deutsch: wenn ... dann ... sonst

Wenn diese Bedingung erfüllt ist...

```
if test $1 = "eins"
then
  echo "$1 ist gleich eins"
else
  echo "$1 ist ungleich eins"
fi
```

dann mache dies
(Bedingung erfüllt)

sonst (nicht erfüllt) mache dies

Bedingte Ausführung (2)

Beispiel: hsort und hsort2 zusammenfassen

```
#!/bin/bash

if test $# = 1          1 Argument?
then                    ja, Aufruf wie hsort
  head -2 $1 ; tail -n +3 $1 | sort
else                    nein, Aufruf wie hsort 2
  head -2 $1 ; tail -n +3 $1 | sort -k $2 -n
fi
```

Bedingte Ausführung (3)

Vorhandensein einer Datei als Bedingung (exists):

```
if test -e $1
then
  echo "Die Datei $1 ist vorhanden!"
else
  echo "Schade, die Datei $1 gibt es nicht."
fi
```

Bedingte Ausführung (4)

```
if test ! -e $1
then
  echo "Die Datei $1 ist nicht vorhanden."
  exit 1
fi
```

Negiert die Bedingung („wenn nicht...“)

Bricht das Skript an dieser Stelle ab

```
if test $# = 1;
then
  head -2 $1 ; tail +3 $1 | sort
else
  head -2 $1 ; tail +3 $1 | sort -k $2 -n
fi
```

Schleifen (1)

Ihr erinnert euch?

```
> for i in img*.jpg; do composite
  untertitel.png -geometry +250+550 $i
  neu-$i; done
```

→ jetzt kommt die Auflösung

Schleifen (2)

Schleifen: for i in ... do ... done
(Deutsch: für alle i in ... führe ... aus)

```
#!/bin/bash

for i in *.jpg
do
  echo $i
done
```

Schleifenvariable

Wertebereich der Schleifenvar.

führe diese Zeile(n) für jeden Wert der Schleifenvariable einmal aus

Schleifen (3)

Beim direkten Eingeben der Schleife:
Zeilenumbrüche durch Semikolon ; ersetzen!

```
#!/bin/bash
for i in *.jpg
do
  echo $i
done
```

```
> for i in *.jpg; do echo $i; done
```

basename

basename : Dateiname ohne Pfad und Endung liefern

```
> basename /homes/cg/bild.jpg
bild.jpg
```

```
> basename ~cg/bild.jpg .jpg
bild
```

Umbenennen von Dateiendungen (1)

Problem: **`mv *.JPG *.jpg`** geht nicht!

Ansatz:

```
> basename bild.JPG .JPG
bild
```

Ausgabe eines Programms der Variablen zuweisen:
Programmauruf in $(...) packen

Variable

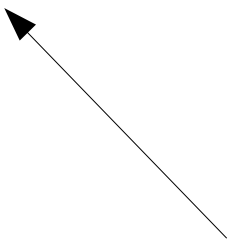
```
> bn=$(basename bild.JPG .JPG)
> echo $bn
bild
```

Umbenennen von Dateiendungen (2)

Erste Version

```
#!/bin/bash

for i in *.JPG
do
  bn=$(basename $i .JPG)
  echo mv $i $bn.jpg
done
```

für alles was auf .JPG endet
führe aus:
.JPG abschneiden
umbenennen



erst mal nur testen!

Umbenennen von Dateiendungen (3)

Zweite Version (Abstraktion: .JPG → $1; .jpg → $2)

```
#!/bin/bash

for i in *.$1                    für alles was auf .JPG endet
do                               führe aus:
  bn=$(basename $i .$1)            .JPG abschneiden
  mv $i $bn.$2                     umbenennen
done
```

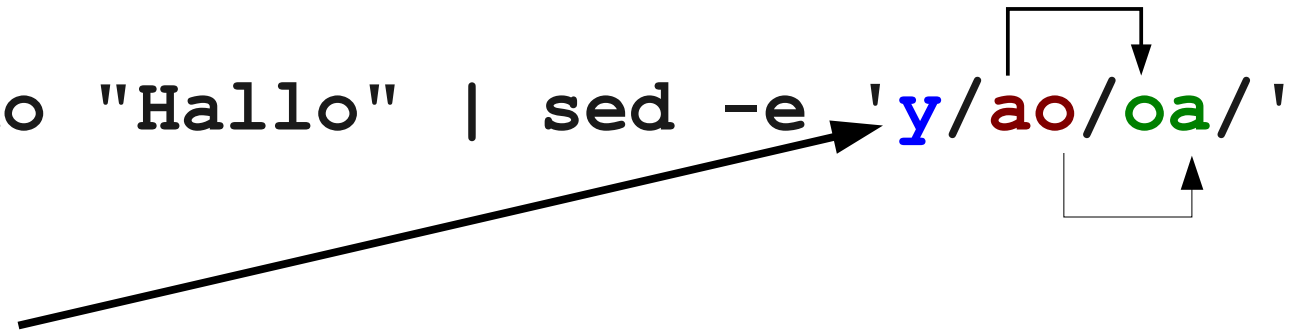
```
> xmv JPG jpg                    Aufrufbeispiel
```

Teile von Worten ersetzen (1)

sed : script editor

„Suchen und Ersetzen“ per Kommandozeile

```
> echo "Hallo" | sed -e 'y/ao/oa/'
Holla
```



„Betriebsart“

y: Buchstaben aus Liste 1 durch diejenigen aus Liste 2 ersetzen

Wechseln zwischen Groß/Kleinbuchst.

Beispiel:

(den folgenden Aufruf in eine Zeile schreiben!)

```
> echo "HALLO" |
  sed -e 'y/ABCDEFGHIJKLMNOPQRSTUVWXYZ/
             abcdefghijklmnopqrstuvwxyz/'

hallo
```

Anwendungsbeispiel 1: “lmv”

```
#!/bin/bash

klein=$(echo $1 | sed -e "y/ABCDEFGHIJKLMN
OPQRSTUVWXYZ/abcdefghijklmnopqrstuvwxyz/")

echo "mv $1 $klein"
mv $1 $klein
```

```
> lmv BILD.JPG
mv BILD.JPG bild.jpg
```

→ praktisch beim Datenaustausch mit Windows/FAT32

Teile von Worten ersetzen (2)

Beispiel für sed-„Betriebsart“ `s`:

```
> echo "img_398.jpg" | sed -e 's/img/bild/'
bild_398.jpg
```

→ ersetzt Vorkommen des ersten Teilwortes
durch das zweite Teilwort

Anwendungsbeispiel 2: „pmv“

```
#!/bin/bash

neu=$(echo $3 | sed -e "s/$1/$2/")

echo "mv $3 $neu"
mv $3 $neu
```

ergibt einen „partiellen Move-Befehl“:

```
> pmv img bild img_2029.jpg
mv img_2029.jpg bild_2029.jpg
```

Zeilen, Wörter und Zeichen zählen

wc : word count

```
> wc gpl.txt
```

zeigt die Anzahl der Zeilen, Wörter und Zeichen an

```
> wc -l gpl.txt
```

zeigt nur die Anzahl der Zeilen in der gpl.txt an

Zeichen austauschen

tr : translate or delete characters

Tauscht Zeichen ähnlich wie **sed** im y-Modus

```
> tr  " "  "\t"
```

Ersetzt Leerzeichen durch Tabs.

```
> tr -s  " "  "\t"
```

Ersetzt mehrere aufeinanderfolgende Leerzeichen

durch ein Tab.

Daten ausschneiden

cut : remove sections from each line of files

Trennt Zeilen an Tabs oder anderen Zeichen in Spalten.

```
> cut -f 2 planeten2.txt
```

wählt zweite Spalte aus

```
> cut -f 2 -d " " eingabe.txt
```

trennt an Leerzeichen und wählt zweite Spalte aus

```
> ls -l | cut -c 31-80
```

wählt Zeichenbereich aus

Dateien herunterladen

wget : The non-interactive network downloader

Läd Dateien von Webservern herunter.

```
> wget http://imgs.xkcd.com/comics/first.png
```

läd einen xkcd-Comic herunter.

```
> wget -O ausgabe.png [URL]
```

speichert die URL als "ausgabe.png"

```
> wget --referer=[REFURL] [URL]
```

behauptet von der Seite REFURL zu kommen

Shell-Skript live programmieren

- Etwas fortgeschrittenes Shell-Skript
- Verwendet bisher vorgestellte Tools
- Google Bildersuche austricksen
- Schön nutzlos :-)

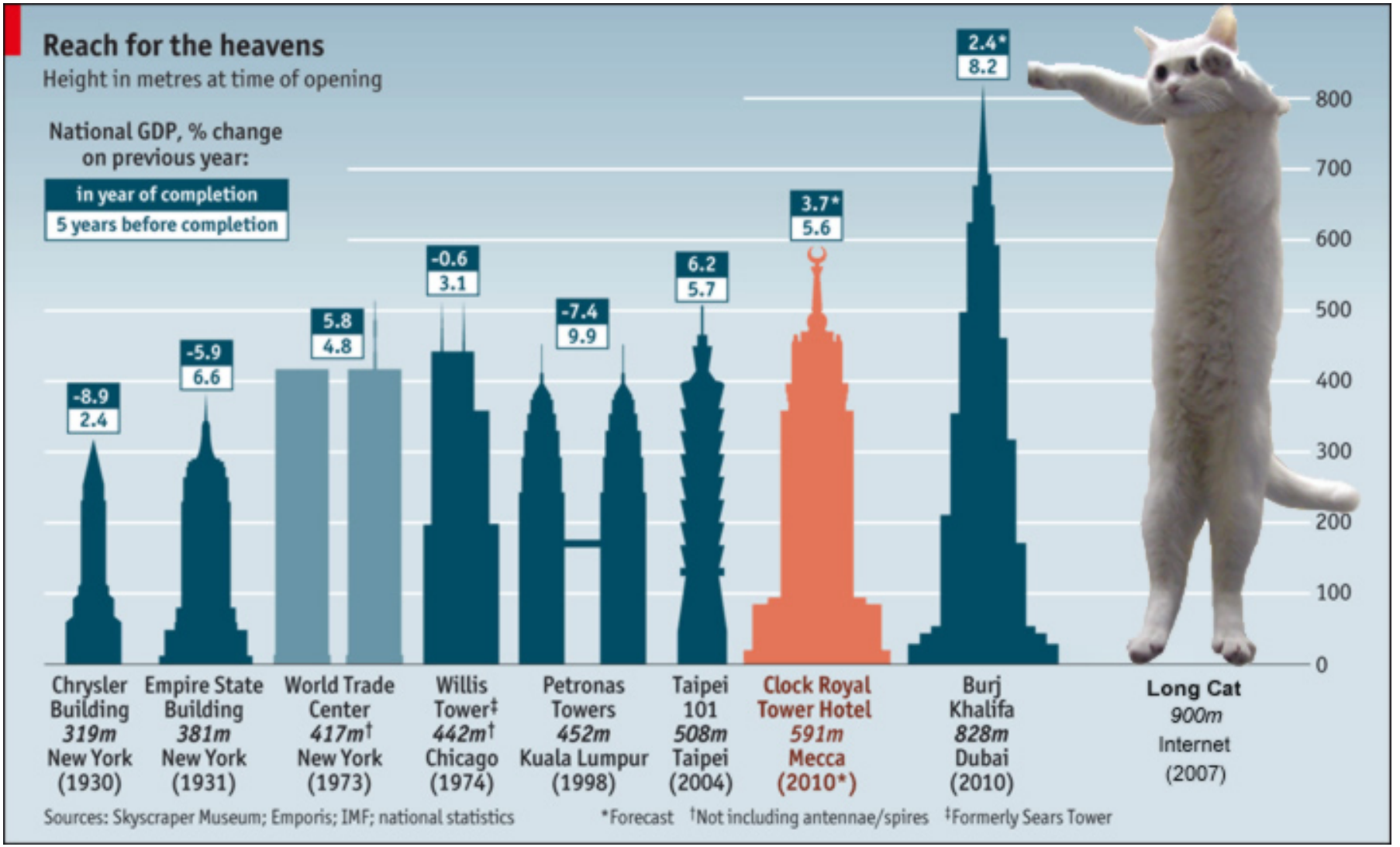
Shell-Skript live programmieren

- Etwas fortgeschrittenes Shell-Skript
- Verwendet bisher vorgestellte Tools
- Google Bildersuche austricksen
- Schön nutzlos :-)

→ Longcat Image Macro Generator

Longcat

Longcat in Aktion



Vorlesung morgen...

Verschiedene Kurzvorträge der Tutoren:

- Screen
- LaTeχ
- Editoren
- Mehr Shell-Scripte
- ...?