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/axel/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/axel/abc.txt.old`)

`${TEST%.*}` → `/home/axel/abc.txt`

`${TEST%%.*}` → `/home/axel/abc`

`${TEST##*/}` → `abc.txt.old`

`${TEST//ab/de}` → `/home/axel/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/axel/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