

Arbeiten in der Unix-Kommandozeile

PeP et al. Toolbox Workshop



PeP et al. e.V.

Physikstudierende und
ehemalige Physikstudierende
der TU Dortmund

2017

```
[ismo@it ~]$ _
```

Was ist das?

Muss das sein?

Ist das nicht völlig veraltet?

Das sieht nicht so schick aus...

Die meisten Geräte basieren auf Unix

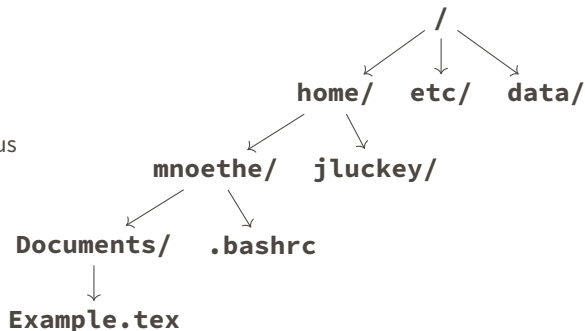
- Server, Cluster, Supercomputer
- Smartphones
- Router, Drucker, ...

Wissenschaftliche Programme werden in der Regel für Unix geschrieben

- Bedienung über Kommandozeile
- Wichtige Programme haben keine GUIs
- z. B. bei der Bachelorarbeit...

- Kommandozeile ist in vielerlei Hinsicht überlegenes Bedienkonzept
 - Die meiste Zeit beim Arbeiten verbringen wir im CLI
- GUIs verstecken die Details
- GUIs sind nicht böse oder schlecht, man muss nur wissen, was dahinter steckt
- In der Kommandozeile ist alles automatisierbar
 - Wenn man etwas zum dritten Mal tut, sollte man ein Skript dafür schreiben
- Arbeiten in GUIs ist nur schwierig reproduzierbar

- bildet *einen* Baum
 - beginnt bei / (root)
 - / trennt Teile eines Pfads
 - auf Groß-/Kleinschreibung achten!
- es gibt ein aktuelles Verzeichnis
- relativer Pfad: vom aktuellen Verzeichnis aus (Kein führender /)
- absoluter Pfad: von / aus
- spezielle Verzeichnisse:
 - das aktuelle Verzeichnis
 - das Oberverzeichnis
 - ~ das Homeverzeichnis
 - das vorherige Verzeichnis



→ Jede Datei hat einen Besitzer und eine Gruppe

→ Lese-, Schreib- und Ausführungsrechte können einzeln vergeben werden

```
-rwxr-xr-x  1 maxnoe maxnoe  177 15. Sep 13:37 toggle-touchpad.sh
-rw-r--r--  1 maxnoe maxnoe  98K 29. Aug 15:52 trigger_arduino.jpg
-rw-r--r--  1 maxnoe maxnoe  410 13. Jul 10:37 trigger.py
drwxr-xr-x 20 maxnoe maxnoe 4.0K 29. Sep 11:19 Uni
```

```
\_/\_/\_/  
U  G  O  
s  r  t  
e  o  h  
r  u  e  
p  r
```

→ r: Lesen, w: Schreiben, x: Ausführen

→ d: Ist Verzeichnis

<code>man <i>topic</i></code>	„manual“: zeigt die Hilfe für ein Programm
<code>pwd</code>	„print working directory“: zeigt das aktuelle Verzeichnis
<code>cd <i>directory</i></code>	„change directory“: wechselt in das angegebene Verzeichnis

ls [*directory*] „list“: zeigt den Inhalt eines Verzeichnisses an

ls -l „long“: zeigt mehr Informationen über Dateien und Verzeichnisse

ls -a „all“: zeigt auch versteckte Dateien (fangen mit . an)

<code>mkdir <i>directory</i></code>	„make directory“: erstellt ein neues Verzeichnis
<code>mkdir -p <i>directory</i></code>	„parent“: erstellt auch alle notwendigen Oberverzeichnisse
<code>touch <i>file</i></code>	erstellt eine leere Datei, falls sie noch nicht existiert ändert Bearbeitungsdatum auf „jetzt“

<code>cp source destination</code>	„copy“: kopiert eine Datei
<code>cp -r source destination</code>	„recursive“: kopiert ein Verzeichnis rekursiv
<code>mv source desination</code>	„move“: verschiebt eine Datei (Umbenennung)
<code>rm file</code>	„remove“: löscht eine Datei (Es gibt keinen Papierkorb!)
<code>rm -r directory</code>	„recursive“: löscht ein Verzeichnis rekursiv
<code>rmdir directory</code>	„remove directory“: löscht ein <i>leeres</i> Verzeichnis

`cat file`

„concatenate“: gibt Inhalt einer (oder mehr) Datei(en) aus

`less file`

(besser als more): wie cat, aber navigabel

`grep pattern file`

g/re/p: sucht in einer Datei nach einem Muster

`grep -i pattern file`

„case insensitive“

`grep -r pattern directory`

„recursive“: suche rekursiv in allen Dateien

`echo message`

gibt einen Text aus

<code>command > file</code>	überschreibt Datei mit Ausgabe
<code>command >> file</code>	fügt Ausgabe einer Datei hinzu
<code>command < file</code>	Datei als Eingabe
<code>command1 command2</code>	Ausgabe als Eingabe (Pipe)

`Ctrl-C` beendet das laufende Programm

`Ctrl-D` EOF (end of file) eingeben, kann Programme beenden

`Ctrl-L` leert den Bildschirm

* wird ersetzt durch alle passenden Dateien

{a,b} bildet alle Kombinationen

Beispiele:

*.log → foo.log bar.log

foo.{tex,pdf} → foo.tex foo.pdf

- Datei enthält Befehle
- Selbe Syntax wie Kommandozeile
- Endung: keine oder `.sh`
- Ausführung:
 - `bash skript`
 - `./skript` (mit Shebang)
- Shebang: erste Zeile enthält Pfad des Interpreters (muss absolut sein)
 - `#!/bin/bash`

- steuern viele Einstellungen und Programme
- Ausgabe mit `echo $NAME`
- wichtiges Beispiel: PATH (auch unter Windows):
 - enthält alle Pfade, in denen nach Programmen gesucht werden soll
 - wird von vorne nach hinten gelesen
 - erster Treffer wird genommen
 - `which program` zeigt den Pfad eines Programms
 - Shebang, das den ersten Treffer im PATH nutzt, statt festem Pfad: `#!/usr/bin/env python`
- Änderung über `export`:
`export PATH=/home/maxnoe/anaconda3/bin:$PATH`

- Einstellungen für viele Programme werden in Textdateien gespeichert
- Üblicherweise versteckte Dateien im HOME-Verzeichnis
- Einstellungen für die Konsole an sich: `.bashrc` (Linux) bzw. `.bash_profile` (Mac)
- Bash-Befehle die beim Start jeder Konsole ausgeführt werden
- Umgebungsvariablen setzen
- Sehr nützlich: `alias`, definiert Alternativform für Befehle

```
alias ll='ls -lh'
alias gits='git status -s'
alias ..='cd ..'
```
- Müssen nach Änderungen neugeladen werden:

```
source ~/.bashrc
```