

Arbeiten in der Unix-Kommandozeile

PeP et al. Toolbox Workshop



PeP et al. e.V.

Physikstudierende und
ehemalige Physikstudierende
der TU Dortmund

2025

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 Bachelor- oder Masterarbeit

- Kommandozeile ist in vielerlei Hinsicht überlegenes Bedienkonzept
 - Die meiste Zeit beim wissenschaftlichen Arbeiten verbringen wir in der Kommandozeile (auch CLI, Command Line Interface)
- GUIs (Graphical User Interface) 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

Terminal-Emulatoren

- Terminals sind im ursprünglichen Sinne Hardware und wurden durch den Personal Computer ersetzt
- Terminal-Emulatoren oder auch Terminalprogramme sind Programme die Terminals auf einem PC emulieren
- Beispiele für graphische Terminal-Emulatoren sind:

Linux

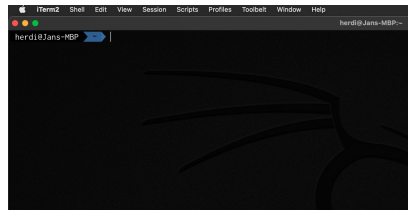
- xterm
- GNOME Terminal
- kitty
- tilix

Windows

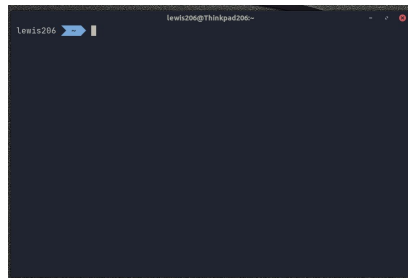
- Windows Konsole
- Windows Terminal

macOS

- iTerm2
- Terminal



iTerm2 in macOS

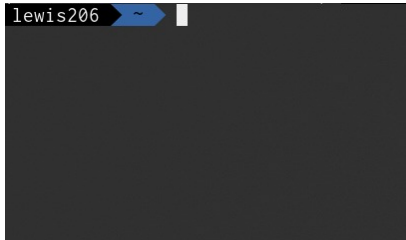


GNOME Terminal in Linux

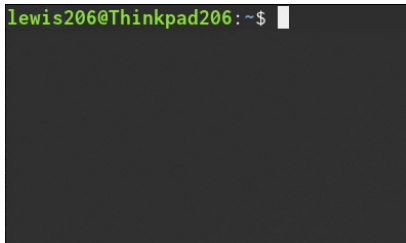
Es gibt verschiedene Tastenkürzel, die sich je nach Terminal-Emulator unterscheiden

iTerm2	Windows Terminal	Gnome-Terminal	Befehl
Enter	Enter	Enter	Befehl ausführen
Ctrl-C	Ctrl-C	Ctrl-C	beendet das laufende Programm
Ctrl-D	Ctrl-D	Ctrl-D	EOF (end of file) eingeben, kann Programme die auf Eingaben warten beenden
Ctrl-L	Ctrl-L	Ctrl-L	leert den Bildschirm
↑ ↓	↑ ↓	↑ ↓	Befehlshistorie durchgehen
Cmd-C	Ctrl-C	Ctrl-Shift-C	Kopieren von Text
Cmd-V	Ctrl-V	Ctrl-Shift-V	Einfügen aus Zwischenablage

- Shells sind die äußerste Ebene des Betriebssystems (OS), daher der Name *shell*
- Funktionen der grafische Shells sind z.B. Desktopumgebungen, Start Menüs und die Taskbar, aber das unterscheidet sich natürlich je nach OS
- Command-Line Shells sind Programme die in den Terminal-Emulatoren laufen und die Verbindung zum OS darstellen
- Typische Command-Line Shells sind bash, zsh, Powershell, cmd.exe, fish, etc.

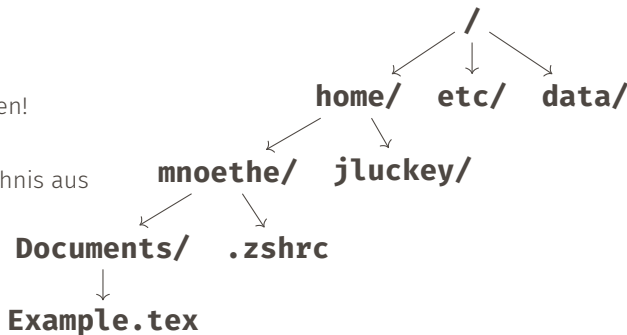


zsh-Shell mit oh-my-zsh agnoster Theme



bash-Shell

- 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 Home-Verzeichnis



<code>ls [directory]</code>	„list“: zeigt den Inhalt eines Verzeichnisses an
<code>ls -l</code>	„long“: zeigt mehr Informationen über Dateien und Verzeichnisse
<code>ls -a</code>	„all“: zeigt auch versteckte Dateien (fangen mit <code>.</code> an)
<code>cd directory</code>	„change directory“: wechselt in das angegebene Verzeichnis
<code>cd -</code>	Wechselt ins vorherige Verzeichnis zurück
<code>pwd</code>	„print working directory“: zeigt das aktuelle Verzeichnis

<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 destination</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

man, cat, less, grep, echo

<code>man topic</code>	„manual“: zeigt die Hilfe für ein Programm
<code>cat file</code>	„concatenate“: gibt Inhalt einer (oder mehr) Datei(en) aus
<code>less file</code>	(besser als more): wie cat , aber navigabel
<code>grep pattern file</code>	g/re/p : sucht in einer Datei nach einem Muster
<code>grep -i pattern file</code>	„case insensitive“
<code>grep -r pattern directory</code>	„recursive“: suche rekursiv in allen Dateien
<code>echo message</code>	gibt einen Text aus

Beispiel: Finde jedes Paket, dass wir in unseren Python-Skripten importieren:

```
$ grep -R --include='*.py' import  
v52_leitungen/scripts/plot_lcrp.py:import matplotlib.pyplot as plt  
v52_leitungen/scripts/plot_lcrp.py:import numpy as np
```

find

Sehr mächtiges Werkzeug, um Dateien und Ordner zu finden, und Befehle auszuführen.

<code>find .</code>	Rekursiv alle Dateien und Ordner im aktuellen Verzeichnis listen
<code>find . -type f</code>	Nur Dateien anzeigen
<code>find . -name '*.py'</code>	Alle Dateien, die mit <code>.py</code> enden
<code>find . -exec <befehl> \;</code>	Befehl für jede gefundene Datei ausführen, wird dabei <code>{}</code> eingesetzt wird dies durch den Dateinamen ersetzt

Beispiel: Schreibe eine Nachricht zu den gefundenen Dateien (hier exemplarisch in einem Verzeichnis, das nur die Datei `example.py` enthält):

```
$ find *.py -exec echo "Found file: {}" \;  
Found file: example.py
```

<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)

Globbering

- * wird ersetzt durch alle passenden Dateien
- {a,b} bildet alle Kombinationen

Beispiele:

```
*.log           →  foo.log bar.log  
foo.{tex,pdf}  →  foo.tex foo.pdf
```

→ 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: read, w: write, x: execute

→ u: user, g: group, o: other, a: all

→ d: Ist Verzeichnis

<code>chmod [options] mode file1 ...</code>	„change mode“: Verändert die klassischen Unix-Dateirechte
<code>chmod a+x beispiel.txt</code>	Beispiel: Fügt bei allen das Recht auf Ausführung hinzu
<code>sudo command</code>	„superuser do“: führt einen Befehl als „root“-User aus
	Achtung: Mit Vorsicht verwenden!

- 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`

- Einstellungen für viele Programme werden in Textdateien gespeichert
- Üblicherweise versteckte Dateien im **HOME**-Verzeichnis
- Einstellungen für die Konsole an sich: **.bashrc**, **.zshrc**, etc.
- 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
```

- 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/.local/texlive/2023/bin/x86_64-linux:$PATH`