Kommandozeile

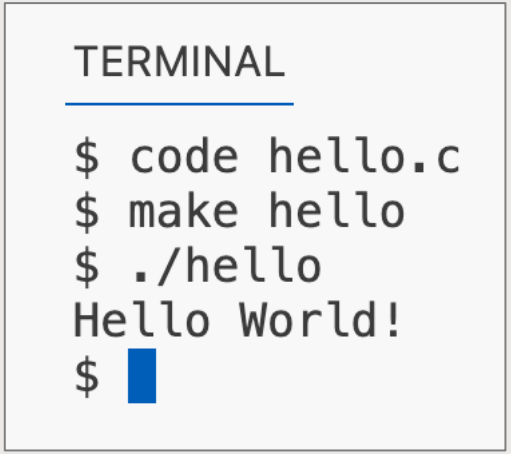
>> Die **Kommandozeile** (command line interface, CLI) ist eine textbasierte Schnittstelle, die eine direkte Interaktion mit dem Betriebssystem eines Computers über ein **Shell-Programm** ermöglicht, auf das in der Regel über eine **Terminal-Anwendung** zugegriffen wird.

Gerade **Linux**, ein weit verbreitetes und frei zugängliches Betriebssystem, bietet viele praktische Werkzeuge, die über die Kommandozeile erreicht werden können. Beim Programmieren und Administrieren von IT-Systemen, aber auch zum Lösen von Alltagsproblemen ist es sehr nützlich.

>> Die in Inf-Einf-B verwendeten CS50 Dev Container führen eine virtuelle Ubuntu-Linux-Maschine aus; Register „Terminal“ in VS Code bietet direkten Zugriff auf die Kommandozeile dieser Linux-Umgebung.

Die gleiche Kommandozeilen ist in Github Codespaces erreichbar, wenn Sie den CS50 Dev Container nutzen.

Das Windows Subsystem für Linux oder macOS bieten ebenfalls Zugriff auf die Kommandozeile.



ls Steht für „list“. Mit diesem Befehl werden alle Dateien und Ordner im aktuellen Verzeichnis angezeigt.

`ls -a` (zeigt alle, auch versteckte Dateien)

`ls -l` (langes Format mit Details)

cd *verz*

Kurz für „change directory“.

Mit diesem Befehl wechseln Sie das aktuelle Verzeichnis und gehen ins Verzeichnis *verz*.

Kurzname für aktuelles Verzeichnis ist `.`

Der Kurzname für das übergeordnete Verzeichnis (**parent**) lautet `..`

Wenn Sie nicht mehr wissen, was der Pfad des aktuellen Verzeichnisses ist, können Sie `pwd` (*print working directory*) eingeben. Meist steht das aktuelle Verzeichnis vor dem **Prompt** ($).

Mit der **Tabulatortaste** (⇆) auf der Tastatur kann man Pfade und Dateinamen bei der Eingabe automatisch vervollständigen lassen.

```
$ cd uebung1
uebung1/ $ ls
uebung1/ $ cd ..
$ ls
test*  test.c  uebung1/  uebung2/
$
```

Dateien und Verzeichnisse

Ein Dateisystem ist in einer baumartigen Struktur organisiert.

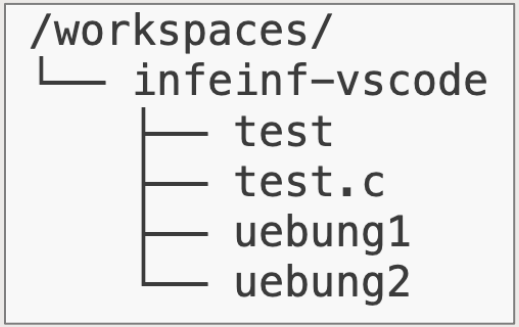
Verzeichnisse (Ordner) können Dateien und andere Verzeichnisse enthalten.

Ein Pfad ist der Weg zu einer bestimmten Datei oder einem bestimmten Verzeichnis.

Absolute Pfade beginnen im Stammverzeichnis (normalerweise mit „/“ gekennzeichnet).

Relative Pfade beginnen im aktuellen Verzeichnis (haben keinen / am Anfang).

Dateinamen enthalten oft Erweiterungen (z. B. .docx, .c, .txt, .py), die den Dateityp angeben.



mkdir *verz*

Dieser Befehl ist die Abkürzung für „make directory“; erstellt ein neues Unterverzeichnis mit dem Namen *verz* im aktuellen Verzeichnis.

```
$ ls
test*  test.c  uebung1/  uebung2/
$ mkdir uebung3
$ ls
test*  test.c  uebung1/  uebung2/  uebung3/
$
```

```
$ cp test.c test2.c
$ cp test.c uebung1
$ ls
test*  test2.c  test.c  uebung1/  uebung2/  uebung3/
$ ls uebung1
test.c
$
```

cp *source destination*

Dieser Befehl ist die Kurzform von „copy“ und ermöglicht es, ein Duplikat der Datei zu erstellen, die man als *source* angibt; gespeichert wird die neue Datei unter *destination*.

Wenn man ganze Verzeichnisse kopieren will, muss man den Befehl leicht abändern:

```
cp -r source_dir dest_dir
```

Das „-r“ steht für rekursiv und weist *cp* an, den ganzen Teilbaum dieses Verzeichnisses zu durchwandern und alles darin zu kopieren (einschließlich aller darin enthaltenen Unterverzeichnisse).

Mit ▲ und ▼ kann man die letzten Befehle durchgehen, verändern und mit *ENTER* noch einmal ausführen.

mv *source destination*

Dieser Befehl ist die Kurzform von „move“ und ermöglicht es, eine Datei in ein anderes Verzeichnis zu verschieben oder sie umzubenennen.

```
$ ls
test*  test2.c  test.c  uebung1/  uebung2/  uebung3/
$ mv test2.c variante.c
$ ls
test*  test.c  uebung1/  uebung2/  uebung3/  variante.c
$ mv test.c uebung1
mv: overwrite 'uebung1/test.c'? y
$ ls uebung1
test.c
$
```

Unter Umständen wird man beim Überschreiben noch einmal gefragt, ob man sicher ist. Gibt man dort etwas anderes als „y“ ein, wird nichts überschrieben.

Man kann auch ganze Verzeichnisse verschieben; bei mv ist dazu kein `-r` erforderlich.

rm *filename*

Abkürzung für „remove“ (entfernen).
Löscht Dateien oder Verzeichnisse.

Vorsicht: es gibt keinen „Papierkorb“
und kein „rückgängig“.

Um ein ganzes Verzeichnis zu löschen,
muss man es rekursiv entfernen:

`rm -r` *`directory`*

Um die Sicherheitsabfrage zu deaktivieren,
kann man bei vielen Befehlen den Parameter
`-f` („force“) hinzufügen.

```
$ ls
test*  uebung1/  uebung2/  uebung3/  variante.c
$ rm variante.c
rm: remove regular file 'variante.c'? y
$ rm uebung3
rm: cannot remove 'uebung3': Is a directory
$ rm -rf uebung3/
$ ls
test*  uebung1/  uebung2/
$
```

Textdateien ausgeben

`cat`: Abkürzung für „concatenate“ (verketten). Zeigt Dateiinhalte an, entweder von einer oder mehrerer Dateien hintereinander.

`head`: Zeigt die ersten Zeilen einer Datei an.

`tail`: Zeigt die letzten Zeilen einer Datei an.

Mit `-n` *`anzahl`* kann man bei *head* und *`tail`* die Anzahl der Zeilen festlegen, standardmäßig werden 10 Zeilen ausgegeben.

```
uebung1/ $ cat hello.c
#include <stdio.h>

int main(void)
{
    printf("Hello World!\n");
    return 0;
}
uebung1/ $ tail -n 3 hello.c
    printf("Hello World!\n");
    return 0;
}
```

Wildcards / Leerzeichen

Ein Stern * steht für beliebig viele Zeichen im Dateinamen:

`ls *.txt` (listet alle .txt Dateien)

Ein Fragezeichen ? steht für genau ein Zeichen:

`cat file??.txt` (zeigt file01.txt, fileA1.txt, etc. an)

Umgang mit Leerzeichen in Dateinamen:

Entweder: Backslash vor Leerzeichen schreiben:
`My\ File.txt`

oder in Anführungszeichen setzen: `"My File.txt"`

Textdateien: Sortieren

Der Befehl `sort` dient zum Sortieren von Zeilen in Textdateien. Gängige Parameter:

`sort file.txt` (alphabetische Sortierung)

```
1
100
2
A
B
Z
```

`sort -n file.txt` (numerische Sortierung)

```
A
B
Z
1
2
100
```

`sort -r file.txt` (umgekehrte Sortierung)

Textdateien: Dubletten behandeln

Der Befehl `uniq` sucht gleiche Zeilen in Textdateien, *die bereits sortiert sind*.

Datei ausgeben, aber ohne aufeinander-folgende gleiche Zeilen zu wiederholen:

```
uniq sorted_file.txt
```

Für jede Zeile ausgeben, wie oft sie vorkommt:

```
uniq -c sorted_file.txt
```

Nur mehrfach auftretende Zeilen ausgeben

```
uniq -d sorted_file.txt
```

Textdateien: einzelne Spalten ausgeben

Wenn in einer Textdatei jede Zeile aus mehreren Teilen besteht, die mit einem Trennzeichen (Delimiter) getrennt sind, kann man mit `cut` bestimmte Teile ausgeben.

Werte in der zweiten Spalte einer kommaseparierten CSV-Datei ausgeben:

```
cut -f2 -d',' data.csv
```

Erstes Wort jeder Zeile ausgeben:

```
cut -f1 -d' ' story.txt
```

```
$ cat data.csv
ID,Ort,Temperatur
1,Bamberg,18
2,Forchheim,21
3,Erlangen,17
$ cut -f3 -d',' data.csv
Temperatur
18
21
17
$
```

Standard-Eingabe (stdin) und -Ausgabe (stdout)

Kanäle für Ein- und Ausgabe auf der Konsole:

stdin: normalerweise Tastatur
stdout: normalerweise Bildschirm
stderr: für Fehlermeldungen

Werden auch in unseren C-Programmen verwendet:

`get_int(...)` liest mittels *stdin*,
`printf(...)` sendet die Ausgabe an *stdout*

Umleitung in Dateien möglich:

`befehl > datei.txt` (überschreibt Datei mit Ausgabe)

`befehl >> datei.txt` (hängt Ausgabe an Datei an)

`befehl < datei.txt` (liest Eingaben aus Datei)

```
$ echo hallo > text.txt
$ echo hallo >> text.txt
$ echo welt >> text.txt
$ sort text.txt | uniq -c
      2 hallo
      1 welt
$
```

Pipes

Pipes (|) verbinden *stdout* eines Befehls mit *stdin* eines anderen.

Dadurch ist es möglich, Befehle hintereinander in einer Pipeline zur Verarbeitung von Daten anzuordnen.

Die Ausgabe des letzten Befehls in der Pipeline wird angezeigt (sofern nicht umgeleitet).

Syntax: `Befehl1 | Befehl2 | Befehl3`

Typische Beispiele:

```
cat zahlen | sort | uniq -c
```

```
cat data.csv | cut -f2 -d' ' | uniq -c
```

EXTRAS IN 3 MINUTEN

FRAGEN – ANTWORTEN – RÄTSEL
UND KURZE ZUSAMMENFASSUNG

F1: Was ist der Unterschied zwischen einem absoluten und einem relativen Pfad?

F2: Welcher Befehl wird verwendet, um den Inhalt einer Datei anzuzeigen?

F3: Was bewirkt der Parameter `-r` beim `cp`-Befehl?

F4: Wie kann man in der Kommandozeile mehrere Dateien auf einmal löschen, die alle mit „temp“ beginnen?

„Ich habe hier ein Programm pins.c, das eine Ausgabe erzeugt, die in etwa so aussieht:“

```
User: 10, PIN: 126531

User: 11, PIN: 123456

...

User 99, PIN: 531216
```

„Ich muss herausfinden, ob alle PINs unterschiedlich sind. Wie kann ich das überprüfen, ohne von Hand die ganze Liste durchgehen zu müssen? Geht das vielleicht auf der Kommandozeile?“

Ja!

```
./pins | cut -d':' -f2 | sort | uniq -c | sort -n
```

Kommandozeile ermöglicht direkte Interaktion mit Betriebssystem

Wildcards (*,?) vereinfachen Arbeit mit mehreren Dateien auf einmal

Wichtige Befehle:
ls, cd, mkdir, cp, mv, rm

Textverarbeitung: sort, uniq und cut für Datenanalyse

Dateisystem:
Baumstruktur mit absoluten (/) und relativen Pfaden

Pipes verbinden Befehle für komplexe Datenverarbeitung