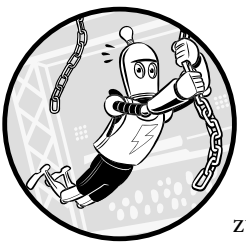
2

Die Einrichtung der Umgebung und die Befehlszeile



Um auf Ihrem Computer Code schreiben zu können, müssen Sie zunächst die *Umgebung einrichten*. Dazu gehört es, alle erforderlichen Tools zu installieren und zu konfigurieren und alle Störungen zu beseitigen, die dabei auftreten mögen. Es gibt keinen festgelegten Einrichtungsprozess, da Programmierer unterschiedliche Computer mit unterschiedlichen Betriebssystemen, Betriebssystemversionen und Versionen des Python-Interpreters nutzen. In diesem Kapitel sehen wir uns daher einige Grundprinzipien zur Verwendung der Befehlszeile, von Umgebungsvariablen und des Dateisystems an.

Es kann lästig erscheinen, diese Prinzipien und Werkzeuge zu lernen. Schließlich wollen Sie Code schreiben und nicht an Konfigurationseinstellungen herumbasteln oder rätselhafte Konsolenbefehle verstehen. Allerdings helfen Ihnen diese Fähigkeiten langfristig, Zeit zu sparen. Wenn Sie Fehlermeldungen ignorieren oder Konfigurationseinstellungen willkürlich ändern, um Ihr System ans Laufen zu bekommen, können Sie Probleme dadurch zwar unterdrücken, aber nicht lösen. Nehmen Sie sich jetzt die Zeit, um diese Schwierigkeiten zu verstehen, sodass Sie später nicht wieder auftreten.

Das Dateisystem

Im *Dateisystem* gliedert das Betriebssystem Daten, um sie speichern und wieder abrufen zu können. Eine Datei zeichnet sich durch zwei Haupteigenschaften aus: den *Dateinamen* (der gewöhnlich in einem Wort geschrieben wird) und den *Pfad*, der den Speicherort der Datei auf dem Computer angibt. Beispielsweise hat eine Datei auf meinem Windows-10-Laptop den Namen *project.docx* und den Pfad *C:\Users\Al\Documents*. Der Teil des Dateinamens hinter dem letzten Punkt ist die *Erweiterung* oder *Endung* und gibt den Typ der Datei an. Bei *project.docx* handelt es sich um ein Word-Dokument, und *Users*, *Al* und *Documents* sind die Bezeichnungen von *Ordnern* (oder *Verzeichnissen*). Ordner können Dateien und wiederum andere Ordner enthalten. Beispielsweise befindet sich *project.docx* im Ordner *Documents*, der wiederum im Ordner *Al* steckt, und dieser schließlich liegt im Ordner *Users*. Abbildung 2–1 zeigt diese Ordnerstruktur.

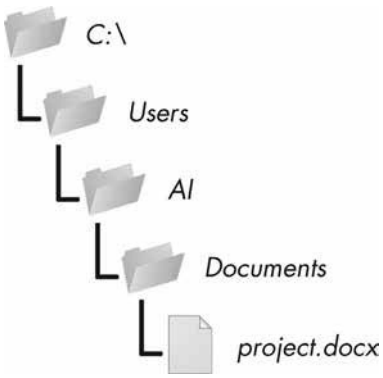


Abb. 2–1 *Eine Datei in einer Hierarchie von Ordnern*

Bei der Angabe *C:* im Pfad handelt es sich um den *Stammordner* (oder *Wurzelordner*, wie er in anderen Betriebssystemen genannt wird), der alle anderen Ordner enthält. In Windows heißt er *C:* und wird auch als »Laufwerk *C:*« bezeichnet, in macOS und Linux heißt er /. In diesem Buch verwende ich bei Pfadangaben den Windows-Stammordner *C:*. Wenn Sie die Beispiele zur interaktiven Shell unter macOS oder Linux ausprobieren wollen, müssen Sie stattdessen / eingeben.

Andere Datenträger, etwa DVD-Laufwerke oder USB-Sticks, werden je nach Betriebssystem unterschiedlich dargestellt. In Windows erscheinen sie als neue, mit den im Alphabet nachfolgenden Buchstaben bezeichnete Stammordner, also etwa *D:* und *E:*. In macOS werden sie als neue Ordner innerhalb des Ordners */Volumes* angezeigt, und in Linux als neue Ordner innerhalb von */mnt* (»mount«). Beachten Sie, dass Linux bei Ordner- und Dateinamen zwischen Groß- und Kleinschreibung unterscheidet, Windows und macOS aber nicht.

Pfade in Python

In Windows werden Ordner- und Dateinamen durch Backslashes (\) getrennt, in macOS und Linux durch normale Schrägstriche (/). Anstatt Ihren Code in zwei Versionen zu schreiben, damit Ihre Python-Skripte plattformübergreifend funktionieren, können Sie das Modul `pathlib` und den Operator / verwenden.

Um `pathlib` zu importieren, verwenden Sie gewöhnlich die Anweisung `from pathlib import Path`. Dadurch können Sie in Ihrem Code einfach `Path` statt `pathlib.Path` schreiben, was sehr praktisch ist, da es sich bei `Path` um die am häufigsten verwendete Klasse dieses Moduls handelt. Wenn Sie den String eines Ordner- oder Dateinamens an `Path()` übergeben, wird ein `Path`-Objekt für diesen Namen erstellt. Solange das Objekt am Anfang eines Ausdrucks ein `Path`-Objekt ist, können Sie zum Verknüpfen der `Path`-Objekte und Strings darin den Operator / verwenden. Geben Sie Folgendes in die interaktive Shell ein:

```
>>> from pathlib import Path
>>> Path('spam') / 'bacon' / 'eggs'
WindowsPath('spam/bacon/eggs')
>>> Path('spam') / Path('bacon/eggs')
WindowsPath('spam/bacon/eggs')
>>> Path('spam') / Path('bacon', 'eggs')
WindowsPath('spam/bacon/eggs')
```

Da ich diesen Code auf einem Windows-Rechner ausgeführt habe, hat `Path()` hier `WindowsPath`-Objekte zurückgegeben. Auf macOS- und Linux-Computern dagegen werden `PosixPath`-Objekte zurückgegeben. (POSIX ist ein Standard für Unix-artige Betriebssysteme, den wir in diesem Buch nicht behandeln werden.) Für unsere Zwecke gibt es keinen Unterschied zwischen diesen beiden Typen.

`Path`-Objekte können Sie an jegliche Funktionen aus der Python-Standardbibliothek übergeben, die einen Dateinamen erwarten. Beispielsweise ist der Funktionsaufruf `open(Path('C:\\') / 'Users' / 'Al' / 'Desktop' / 'spam.py')` gleichwertig mit `open(r'C:\Users\Al\Desktop\spam.py')`.

Das Benutzerverzeichnis

Alle Benutzer haben einen *Benutzerordner* oder ein *Benutzerverzeichnis* für ihre Dateien auf dem Computer. Das `Path`-Objekt für diesen Ordner können Sie mit `Path.home()` abrufen:

```
>>> Path.home()
WindowsPath('C:/Users/Al')
```

Die Benutzerverzeichnisse befinden sich an einem festen Speicherort, der vom Betriebssystem abhängt:

- In Windows befinden sich die Benutzerverzeichnisse in *C:\Users*.
- In macOS befinden sich die Benutzerverzeichnisse in */Users*.
- In Linux befinden sich die Benutzerverzeichnisse meistens in */home*.

Sehr wahrscheinlich werden Ihre Skripte die Berechtigung haben, Dateien in Ihrem Benutzerverzeichnis zu lesen und zu schreiben. Daher ist dieser Ordner der ideale Platz, um die Dateien zu speichern, mit denen Ihre Python-Programme arbeiten.

Das aktuelle Arbeitsverzeichnis

Jedes Programm, das auf Ihrem Rechner läuft, hat ein *aktuelles Arbeitsverzeichnis*. Wenn ein Dateiname oder Pfad nicht mit dem Stammordner beginnt, können Sie davon ausgehen, dass er sich in diesem Arbeitsverzeichnis befindet. Normalerweise ist »Ordner« die moderne Bezeichnung für ein Verzeichnis, doch in diesem Fall ist »Arbeitsverzeichnis« der übliche Begriff und nicht etwa »Arbeitsordner«.

Das aktuelle Arbeitsverzeichnis können Sie mit der Funktion `Path.cwd()` als `Path`-Objekt abrufen (wobei *cwd* für »current working directory« steht) und mit `os.chdir()` ändern. Probieren Sie das wie folgt in der interaktiven Shell aus:

```
>>> from pathlib import Path
>>> import os
>>> Path.cwd()      ❶
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python38')
>>> os.chdir('C:\\Windows\\System32')      ❷
>>> Path.cwd()
WindowsPath('C:/Windows/System32')
```

Hier war das aktuelle Arbeitsverzeichnis zu Anfang auf *C:\Users\Al\AppData\Local\Programs\Python\Python38* gesetzt ❶, sodass der Dateiname *project.docx* auf *C:\Users\Al\AppData\Local\Programs\Python\Python38\project.docx* verweist. Nach der Änderung des Arbeitsverzeichnisses in *C:\Windows\System32* ❷ verweist derselbe Dateiname auf *C:\Windows\System32\project.docx*.

Wenn Sie versuchen, ein Verzeichnis zu ändern, das nicht existiert, zeigt Python eine Fehlermeldung an:

```
>>> os.chdir('C:/ThisFolderDoesNotExist')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [WinError 2] The system cannot find the file specified:
'C:/ThisFolderDoesNotExist'
```

Die Funktion `os.getcwd()` im Modul `os` stellt eine ältere Möglichkeit dar, um das aktuelle Arbeitsverzeichnis als String abzurufen.

Absolute und relative Pfade

Es gibt zwei Möglichkeiten, um einen Dateipfad anzugeben:

- Als absoluter Pfad, der mit dem Stammordner beginnt
- Als relativer Pfad in Bezug auf das aktuelle Arbeitsverzeichnis

Darüber hinaus gibt es noch zwei besondere Ordner, die nur durch einen einzelnen Punkt (`.`) bzw. zwei Punkte (`..`) bezeichnet werden. Dabei handelt es sich nicht um echte Ordner, sondern um besondere Bezeichnungen, die Sie in einem Pfad verwenden können. Der einzelne Punkt steht für »dieses Verzeichnis« und die beiden Punkte für »der Elternordner«.

Abbildung 2–2 zeigt ein Beispiel mit mehreren Ordnern und Dateien. Wenn das aktuelle Arbeitsverzeichnis auf *C:\bacon* gesetzt ist, lauten die relativen Pfade für die anderen Ordner und Dateien wie in der Abbildung angegeben.

Die Angabe .\ am Anfang eines relativen Pfads ist optional. So beziehen sich beispielsweise *.\spam.txt* und `spam.txt` beide auf dieselbe Datei.

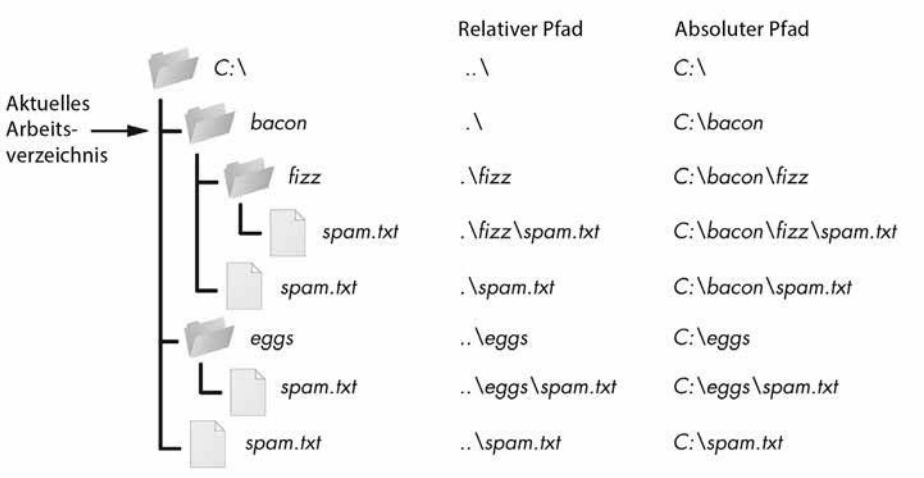


Abb. 2–2 *Relative Pfade für Ordner und Dateien im Arbeitsverzeichnis* C:\bacon

Programme und Prozesse

Ein *Programm* ist eine ausführbare Softwareanwendung, z. B. ein Webbrowser, ein Tabellenkalkulations- oder Textverarbeitungsprogramm, und ein *Prozess* ist eine laufende Instanz eines Programms. Abbildung 2–3 zeigt fünf laufende Prozesse desselben Taschenrechnerprogramms.

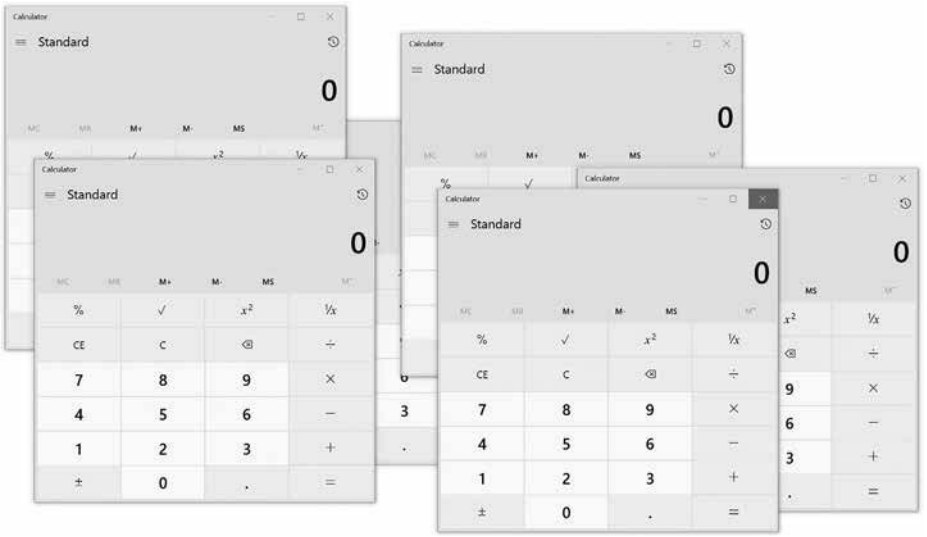


Abb. 2–3 *Das Taschenrechnerprogramm läuft in mehreren voneinander getrennten Prozessen.*

Prozesse werden unabhängig voneinander ausgeführt, selbst diejenigen desselben Programms. Wenn Sie beispielsweise gleichzeitig mehrere Instanzen eines Python-Programms ausführen, können die Variablen in den einzelnen Prozessen unterschiedliche Werte aufweisen. Jeder Prozess hat auch sein eigenes Arbeitsverzeichnis und seine eigenen Umgebungsvariablen. Eine Befehlszeile führt grundsätzlich immer nur einen Prozess auf einmal aus (wobei Sie jedoch mehrere Befehlszeilen gleichzeitig geöffnet haben können).

Alle Betriebssysteme bieten eine Möglichkeit, um sich eine Liste der laufenden Prozesse anzusehen. In Windows müssen Sie dazu Strg + ⇧ + Esc drücken, um den Task-Manager aufzurufen. In macOS führen Sie *Programme* > *Dienstprogramme* > *Aktivitätsanzeige* aus. In Ubuntu Linux öffnen Sie mit Strg + Alt + Entf eine Anwendung, die ebenfalls als Task-Manager bezeichnet wird. Mit diesen Verwaltungsprogrammen können Sie auch laufende Prozesse beenden, die nicht mehr reagieren.

Die Befehlszeile

Die *Befehlszeile* ist ein textgestütztes Programm, in dem Sie Befehle eingeben können, um mit dem Betriebssystem zu arbeiten und andere Programme auszuführen. Sie wird auch als Kommandozeile, Terminal, Shell, Konsole oder CLI (Command Line Interface) bezeichnet und stellt eine Alternative zur grafischen Benutzerschnittstelle (Graphical User Interface, GUI) dar. Eine GUI bietet Ihnen für die Arbeit mit dem Computer mehr als nur Textangabe und gibt optische Anhaltspunkte, die Ihnen die Arbeit erleichtern. Für die meisten Benutzer stellt die Befehlszeile eine Funktion für Fortgeschrittene dar, die sie niemals anrühren. Was die Befehlszeile so abschreckend macht, ist der völlige Mangel an Hinweisen zu ihrer Verwendung. In einer GUI werden Ihnen beispielsweise Schaltflächen angezeigt, sodass Sie wissen, wo Sie zu klicken haben, aber ein leeres Terminalfenster sagt Ihnen nicht, was Sie eingeben müssen.

Es gibt jedoch gute Gründe, um sich mit der Befehlszeile vertraut zu machen. Erstens ist es für die Einrichtung der Umgebung oft erforderlich, die Befehlszeile statt eines grafischen Fensters zu nutzen. Zweitens geht es viel schneller, Befehle einzugeben, als in Fenstern herumzuklicken. Bei Befehlen in Textform wird auch deutlicher, was sie bewirken, als dabei, irgendein Symbol auf ein anderes zu ziehen. Sie lassen sich überdies besser automatisieren, da es möglich ist, mehrere einzelne Befehle zu einem Skript zu kombinieren, mit dem Sie dann anspruchsvolle Aufgaben ausführen können.

Das Befehlszeilenprogramm befindet sich in einer ausführbaren Datei auf Ihrem Computer. In diesem Zusammenhang wird sie oft als Shell oder Shellprogramm bezeichnet. Wenn Sie dieses Programm ausführen, erscheint das Terminalfenster.

- In Windows befindet sich das Shellprogramm unter *C:\Windows\System32\cmd.exe*.
- In macOS befindet sich das Shellprogramm in */bin/bash*.
- In Ubuntu Linux befindet sich das Shellprogramm ebenfalls in */bin/bash*.

Im Laufe der Zeit haben Programmierer viele Shellprogramme für das Betriebssystem Unix entwickelt, darunter die Bourne-Shell (in der ausführbaren Datei *sh*) und später die Bourne-Again-Shell (*Bash*). Linux verwendet standardmäßig Bash, macOS dagegen ab Version Catalina die ähnliche Zsh- oder Z-Shell. Aufgrund der völlig anderen Entwicklungsgeschichte nutzt Windows eine eigene Shell, die als *Eingabeaufforderung* bezeichnet wird. All diese verschiedenen Programme machen aber im Grunde genommen das Gleiche: Sie zeigen ein Terminalfenster mit einer textgestützten Befehlszeile an, in der Sie Befehle eingeben und Programme ausführen können.

In diesem Abschnitt lernen Sie einige der Grundprinzipien und gängigen Befehle der Befehlszeile kennen. Sie könnten zwar eine große Menge der kryptischen Befehle lernen, um zu einem echten Computermagier zu werden, aber etwa ein Dutzend Befehle reichen schon aus, um die meisten Probleme zu lösen. Die genauen Namen der Befehle unterscheiden sich ein wenig von einem Betriebssystem zum anderen, aber die Grundprinzipien sind überall gleich.

Ein Terminalfenster öffnen

Um ein Terminalfenster zu öffnen, gehen Sie wie folgt vor:

- In Windows klicken Sie auf die Startschaltfläche, geben **cmd** ein und drücken die ↵-Taste.
- In macOS klicken Sie auf das Spotlight-Symbol oben rechts, geben **Terminal** ein und drücken die ↵-Taste.
- In Ubuntu-Linux drücken Sie die WIN-Taste, um Dash aufzurufen, geben **Terminal** ein und drücken die ↵-Taste. Alternativ können Sie auch das Tastenkürzel Strg + Alt + T verwenden.

Ähnlich wie die interaktive Shell, die die Eingabeaufforderung >>> anzeigt, gibt ein Terminal eine *Shell-Eingabeaufforderung* zur Eingabe von Befehlen aus. In Windows besteht diese Eingabeaufforderung aus dem vollständigen Pfad zu dem Ordner, in dem Sie sich gerade befinden:

```
C:\Users\Al>geben Sie hier Ihre Befehle ein
```

In macOS besteht die Eingabeaufforderung aus dem Computernamen, einem Doppelpunkt und dem aktuellen Arbeitsverzeichnis, wobei der Benutzerordner durch eine Tilde (~) dargestellt ist. Darauf folgen Ihr Benutzername und ein Dollarzeichen:

```
Als-MacBook-Pro:~ al$ geben Sie hier Ihre Befehle ein
```

Die Eingabeaufforderung von Ubuntu Linux ähnelt der von macOS, beginnt aber mit dem Benutzernamen und dem At-Symbol:

```
al@al-VirtualBox:~$ geben Sie hier Ihre Befehle ein
```

In vielen Büchern und Anleitungen wird die Eingabeaufforderung der Befehlszeile der Einfachheit halber als $ wiedergegeben. Es ist zwar möglich, das Erscheinungsbild der Eingabeaufforderung zu ändern, aber damit wollen wir uns in diesem Buch nicht belasten.

Programme an der Befehlszeile ausführen

Um ein Programm oder einen Befehl auszuführen, müssen Sie seinen Namen in der Befehlszeile eingeben. Sehen wir uns das am Beispiel des Taschenrechnerprogramms an, das zum Lieferumfang des Betriebssystems gehört.

Geben Sie Folgendes in die Befehlszeile ein:

- In Windows: **calc.exe**
- In macOS: **open -a Calculator** (dadurch wird das Programm *open* ausgeführt, das wiederum das Taschenrechnerprogramm ausführt)
- In Linux: **gnome-calculator**

Linux unterscheidet bei Programm- und Befehlsnamen zwischen Groß- und Kleinschreibung, Windows und macOS dagegen nicht. Während Sie also in Linux unbedingt **gnome-calculator** eingeben müssen, können Sie in Windows auch **Calc.exe** oder in macOS **OPEN -a Calculator** schreiben.

Durch die Eingabe des Programmnamens in der Befehlszeile wird das Taschenrechnerprogramm genauso ausgeführt wie über das Startmenü, den Finder oder Dash. Die Programmnamen funktionieren wie Befehle, da sich die Programme *calc.exe*, *open* und *gnome-calculator* in Ordnern befinden, die in der Umgebungsvariablen `PATH` verzeichnet sind. Wenn Sie einen Programmnamen in der Befehlszeile eingeben, prüft die Shell, ob es ein Programm mit dieser Bezeichnung in einem der in `PATH` aufgeführten Ordner gibt. (Mehr darüber erfahren Sie im Abschnitt »PATH und andere Umgebungsvariablen« auf Seite 44.) In Windows sucht die Shell das Programm erst im aktuellen Arbeitsverzeichnis (das in der Eingabeaufforderung angegeben ist), bevor sie die in `PATH` genannten Ordner prüft. Um die Befehlszeile in macOS und Linux anzuweisen, zuerst im aktuellen Arbeitsverzeichnis nachzusehen, müssen Sie dem Dateinamen `./` voranstellen.

Befindet sich das Programm nicht in einem der in `PATH` aufgeführten Ordner, haben Sie die folgenden beiden Möglichkeiten:

- Wechseln Sie mit `cd` zu dem Ordner, in dem sich das Programm befindet, und geben Sie dort den Programmnamen ein. Dazu können Sie beispielsweise die folgenden Befehle verwenden:

  ```
  cd C:\Windows\System32
  calc.exe
  ```

- Geben Sie den kompletten Pfad zur ausführbaren Datei des Programms ein, also etwa *C:\Windows\System32\calc.exe* statt einfach nur *calc.exe*.

Die Angabe der Erweiterung *.exe* oder *.bat* ist in Windows optional. Statt **calc.exe** können Sie genauso gut auch **calc** schreiben. In macOS und Linux dagegen haben ausführbare Dateien oft gar keine besondere Endung, sondern zeichnen sich durch die Berechtigungen von ausführbaren Dateien aus. Mehr darüber erfahren Sie im Abschnitt »Python-Programme außerhalb der Befehlszeile ausführen« auf Seite 49.

Befehlszeilenargumente

Befehlszeilenargumente sind Textelemente, die Sie hinter dem Befehlsnamen angeben. Ähnliche wie Argumente, die Sie in einem Python-Funktionsaufruf übergeben, stellen sie besondere Optionen oder zusätzliche Anweisungen für den Befehl bereit. Wenn Sie beispielsweise den Befehl `cd C:\Users` eingeben, ist `C:\Users` ein Argument, das dem Befehl `cd` mitteilt, zu welchem Ordner er wechseln soll. Bei dem Befehl `python yourScript.py` ist `yourScript.py` das Argument, das angibt, in welcher Datei `python` nach den auszuführenden Anweisungen suchen soll.

Bei den *Befehlszeilenoptionen* (die auch als Flags, Schalter oder einfach Optionen bezeichnet werden) handelt es sich um Befehlszeilenargumente in Form einzelner Buchstaben oder kurzer Wörter. In Windows werden Sie oft mit einem Schrägstrich (`/`) eingeleitet, in macOS und Linux mit einem oder zwei Bindestrichen (`-` oder `--`). Bei dem macOS-Befehl `open -a Calculator` haben Sie bereits die Option `-a` kennengelernt. In macOS und Linux wird bei den Optionen oft zwischen Groß- und Kleinschreibung unterschieden, in Windows nicht. Wenn Sie mehrere Befehlszeilenoptionen angeben, müssen Sie sie in Windows durch Leerzeichen trennen.

Ordner- und Dateinamen werden häufig als Befehlszeilenargumente angegeben. Wenn sich innerhalb eines solchen Namens ein Leerzeichen befinden sollte, müssen Sie ihn in doppelte Anführungszeichen einschließen. Nehmen wir beispielsweise an, Sie wollen zum Ordner *Vacation Photos* wechseln. Bei der Eingabe `cd Vacation Photos` würde die Befehlszeile jedoch davon ausgehen, dass Sie zwei Argumente übergeben, nämlich `Vacation` und `Photos`. Stattdessen müssen Sie `cd "Vacation Photos"` schreiben:

```
C:\Users\Al>cd "Vacation Photos"
C:\Users\Al\Vacation Photos>
```

Eine praktische Option für viele Befehle ist `--help` in macOS und Linux bzw. `/?` in Windows. Dadurch werden Informationen über den Befehl angezeigt. Wenn Sie in Windows beispielsweise `cd /?` eingeben, teilt Ihnen die Shell mit, was der Befehl `cd` macht und welche Argumente und Optionen Sie dafür verwenden können:

```
C:\Users\Al>cd /?
Displays the name of or changes the current directory.

CHDIR [/D] [drive:][path]
CHDIR [..]
CD [/D] [drive:][path]
CD [..]

  .. Specifies that you want to change to the parent directory.

Type CD drive: to display the current directory in the specified drive.
Type CD without parameters to display the current drive and directory.

Use the /D switch to change current drive in addition to changing current
directory for a drive.
-- schnipp --
```

Diesen Informationen können Sie entnehmen, dass der Windows-Befehl `cd` auch unter der Bezeichnung `chdir` zu erreichen ist (wobei aber kaum jemand `chdir` schreiben wird, wenn `cd` reicht). In eckigen Klammern sind die optionalen Argumente angegeben. Beispielsweise bedeutet `CD [/D] [drive:][path]`, dass Sie mit der Option `/D` ein Laufwerk oder einen Pfad angeben können.

Während `/?` und `--help` für erfahrene Benutzer zum Nachschlagen und als Gedächtnisstütze sehr wertvoll sind, wirken die gebotenen Erklärungen leider oft ziemlich undurchsichtig und eignen sich daher nicht so gut für Einsteiger. Neulinge sind mit einem Buch oder einem Webtutorial besser bedient, etwa mit *The Linux Command Line* (2. Ausgabe, 2019) von William Shotts, *Linux Basics for Hackers* (2018) von OccupyTheWeb oder *PowerShell for Sysadmins* (2020) von Adam Bertram (alle erschienen bei No Starch Press).

Python-Code mit –c an der Befehlszeile ausführen

Wenn Sie nur wenige Python-Befehle einmalig ausführen wollen, können Sie hinter `python.exe` in Windows bzw. `python3` in macOS und Linux den Schalter `-c` und dahinter den auszuführenden Code in doppelten Anführungszeichen angeben. Betrachten Sie dazu das folgende Beispiel:

```
C:\Users\Al>python -c "print('Hello, world')"
Hello, world
```

Der Schalter `-c` ist praktisch, wenn Sie sich das Ergebnis einer einzelnen Python-Anweisung ansehen möchten, ohne erst die interaktive Shell aufrufen zu müssen. Beispielsweise können Sie sich schnell die Ausgabe der Funktion `help()` ansehen und dann wieder zur Befehlszeile zurückkehren:

```
C:\Users\Al>python -c "help(len)"
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.

C:\Users\Al>
```

Python-Programme an der Befehlszeile ausführen

Python-Programme sind Textdateien mit der Erweiterung `.py`. Es handelt sich dabei nicht um ausführbare Dateien. Der Python-Interpreter liest sie und führt die darin enthaltenen Python-Anweisungen aus. Die ausführbare Datei dieses Interpreters heißt in Windows *python.exe* und in macOS und Linux *python3* (während dort die ursprüngliche Datei *python* den Interpreter für Python 2 enthält). Wenn Sie den Befehl `python yourScript.py` bzw. `python3 yourScript.py` geben, werden die in der Datei *IhrSkript.py* gespeicherten Python-Anweisungen ausgeführt.

Py.exe

In Windows installiert Python das Programm *py.exe* im Ordner *C:\Windows*. Es ist identisch mit *python.exe*, akzeptiert aber ein zusätzliches Befehlszeilenargument, mit dem Sie jede beliebige Python-Version ausführen können, die auf Ihrem Rechner installiert ist. Da *C:\Windows* in der Umgebungsvariablen `PATH` aufgeführt ist, können Sie den Befehl `py` von jedem beliebigen Ordner aus ausführen. Sind bei Ihnen mehrere Python-Versionen installiert, so führt `py` standardmäßig die neueste davon aus. Außerdem können Sie das Befehlszeilenargument `-3` oder `-2` übergeben, um die neueste Version von Python 3 bzw. Python 2 zu verwenden. Eine weitere Möglichkeit besteht darin, eine genaue Versionsnummer anzugeben, z. B. `-3.6` oder `-2.7`. Hinter diesem Versionsschalter können Sie dann die gleichen Befehlszeilenargumente verwenden wie bei *python.exe*. Probieren Sie das aus, indem Sie an der Windows-Eingabeaufforderung Folgendes eingeben:

```
C:\Users\Al>py -3.6 -c "import sys;print(sys.version)"
3.6.6 (v3.6.6:4cf1f54eb7, Jun 27 2018, 03:37:03) [MSC v.1900 64 bit (AMD64)]

C:\Users\Al>py -2.7
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Das Programm *py.exe* ist praktisch, wenn auf Ihrem Windows-Computer mehrere Python-Versionen installiert sind und Sie eine bestimmte davon ausführen möchten.