

A Externe, lokale Befehlsausführung aus Python

Repository: Ordner `py_adminscripting`.

A.1 Systemprogrammierung in Python

Unter Systemprogrammierung versteht man die Programmierung von Softwarekomponenten, die Teil des Betriebssystems sind oder die eng mit dem Betriebssystem bzw. mit der darunter liegenden Hardware kommunizieren müssen. Systemnahe Software, wie beispielsweise das `sys`-Modul in Python, fungiert als Abstraktionsschicht zwischen der Anwendung, dem Python-Programm, und dem Betriebssystem. Mit Hilfe dieser Abstraktionsschicht lassen sich Applikationen plattformunabhängig implementieren, auch wenn sie direkt auf Betriebssystemfunktionalitäten zugreifen.¹

Wir haben bereits im dritten Jahrgang in Systemtechnik / Betriebssysteme gelernt, wie man unter Linux und Windows Scripts programmiert. Insbesondere die *Powershell* stellt eine äußerst mächtige, objektorientierte Scripting-Umgebung zur Verfügung. Inzwischen existieren auch für Python Bibliotheken, mit denen man ebenfalls sehr effiziente Scripts zur Systemadministration programmieren kann.

A.2 pathlib-Modul

Das PythonModul `pathlib` wird verwendet, um Dateipfade auf eine objektorientierte Weise zu manipulieren und zu verwalten und wird deshalb meist gegenüber dem älteren `os.path`-Modul bevorzugt.

Siehe dazu die Dokumentation bzw. im beiliegendem Cheatsheet:

- <https://realpython.com/python-pathlib/#python-pathlib-examples>
- <https://docs.python.org/3/library/pathlib.html>

Darin enthalten ist die Klasse `Path`, mit der sich die verschiedenen Teile eines Datei- oder Verzeichnispfades einfach extrahieren lassen:²

```
from pathlib import Path
p = Path(r"C:\Users\junioradmin\Pycharm\py_adminscripting\get_path_components.py")
```

- `p.name`: Der Dateiname ohne Verzeichnis
- `p.stem`: Der Dateiname ohne die Dateierweiterung
- `p.suffix`: Die Dateierweiterung
- `p.anchor`: Der Teil des Pfades vor den Verzeichnissen
- `p.parent`: Das Verzeichnis, das die Datei enthält bzw. das übergeordnete Verzeichnis (wenn der Pfad ein Verzeichnis ist)

- (1) Schreibe die Datei `get_path_components.py`, das den absoluten Pfad einer Datei einliest (`input()`) und die oben gelisteten Verzeichnisteile ausgibt. Außerdem soll das übergeordnete Verzeichnis des Ordners (in dem sich die Datei befindet) ausgegeben werden. Falls dieses nicht existiert, gibt das Programm stattdessen aus: *Parent directory does not exist!*.
- (2) Implementiere `find_files_by_ext.py`: Das Programm liest über die Tastatur einen Pfad und eine Dateinamenserweiterung ein und durchsucht rekursiv nach allen Dateien mit dieser Extension und gibt alle gefundenen Dateien mit ihrem absoluten Pfad aus.

Zu diesen Zweck gibt es die Methoden `iterdir()`, welche über alle Dateien im gegebenen Verzeichnis iteriert.

```
from pathlib import Path
from collections import Counter
print(Counter(path.suffix for path in Path.cwd().iterdir()))
```

Möchte man Dateien suchen, deren Namen einem bestimmten Suchmuster entsprechen, bieten sich die Methoden `glob()` und das rekursive `rglob()` an:

¹Quelle: https://www.python-kurs.eu/os_modul_shell.php

²siehe <https://realpython.com/python-pathlib/#picking-out-components-of-a-path>

```
def tree(directory:Path, extension:str):
    print(f"+ {directory}")
    for path in sorted(directory.rglob("*.py")):
        depth = len(path.relative_to(directory).parts)
        spacer = " " * depth
        print(f"{spacer}+ {path.name}")
```

A.3 Fotos archivieren: Python-Bibliothek shutil

Viele Digitalkameras und Smartphones legen ihre Fotos in einem einzigen, unübersichtlichen Ordner (meist unter DCIM = *D*igital *C*amera *I*mage) ab. Dabei werden im Dateinamen häufig Aufnahmedatum und -uhrzeit codiert. Beispiel: 20231113_083245-1.jpg wurde am 13. November 2023 um 08:32 und 45 Sekunden erstellt.

- (3) Schreibe das Programm `copy-dicm.py`, das zunächst die Verzeichnisse *SourcePath* und *DestinationPath* (via Tastatur) einliest. Anschließend kopiert das Programm alle Dateien mit der Extension `.jpg` aus dem Ordner *SourcePath* in eine datumsbasierte Ordnerstruktur unter dem Verzeichnis *DestinationPath*: Zielordner werden – falls erforderlich – automatisch erstellt. Tipp: Zum Kopieren von Dateien mit Python-Skripten verwendet man das Modul `shutil`³: siehe <https://docs.python.org/3.14/library/shutil.html#module-shutil>

Beispiel: Im *SourcePath* `X:\DCIM` befinden sich folgende Dateien, die das Kommando in den *DestinationPath* `C:\pic` wie angegeben kopiert:

```
X:\DCIM\20231231_235959.jpg    --> C:\pic\2023\12\31\20231231_235959.jpg
X:\DCIM\20240101_000001.jpg    --> C:\pic\2024\01\01\20240101_000001.jpg
X:\DCIM\20241113_083245juhu.jpg --> C:\pic\2024\11\13\20241113_083245juhu.jpg
X:\DCIM\20241113_083245-2.jpg  --> C:\pic\2024\11\13\20241113_083245-2.jpg
X:\DCIM\20240101_172418.gif    --> nicht kopieren: falsche Extension
X:\DCIM\oje.jpg               --> nicht kopieren - Dateiname hat keinen Zeitstempel
```

A.4 Externe Programme aus Python aufrufen: subprocess-Modul

Python erlaubt es, externe Programme entweder mit `subprocess.run()` oder `os.system()` zu starten. Gegenüber dem `os`-Modul ist das deutlich modernere `subprocess`-Modul vorzuziehen:

- Flexibilität: `subprocess` bietet eine flexiblere und leistungsfähigere Schnittstelle zum Ausführen externer Programme (z.B. Popen-Objekte zur detaillierten Kontrolle über die Ein- und Ausgabe von Prozessen)
- Sicherheit: `subprocess` bietet sicherere Methoden zum Ausführen von Shell-Befehlen, da es die Verwendung von Listen zur Übergabe von Argumenten unterstützt. Dies reduziert das Risiko von Shell-Injection-Angriffen.
- Erweiterte Funktionen: `subprocess` bietet erweiterte Funktionen wie das Weiterleiten von Ein- und Ausgaben zwischen Prozessen, das Festlegen von Zeitlimits für Prozesse und das Abrufen von Rückgabewerten und Fehlercodes.

`subprocess.run` führt einen Befehl *blockierend* aus, dh es wird gewartet, bis der Befehl abgeschlossen ist. Zurückgegeben wird ein `CompletedProcess`-Objekt, das Informationen über den ausgeführten Befehl enthält, einschließlich des Rückgabecodes⁴, der Standardausgabe und der Standardfehlerausgabe.

Die nachfolgende Programme (bitte ausprobieren!) zeigen diesen Sachverhalt. Man kann hier auch sehr schön das blockierende Verhalten von `subprocess.run()` beobachten.

```
# file write_stdout_stderr.py
import sys, time

if __name__ == "__main__":
```

³Ab Python Version 3.14 kann das auch die oben vorgestellte Klasse `Path`.

⁴Der Parameter `check=False` bewirkt, dass im Falle eines Fehlers (d.h. wenn der ausgeführte Befehl einen Rückgabewert ungleich 0 liefert) keine Ausnahme ausgelöst wird.

```
print("Dies ist eine Nachricht auf stdout") # Ausgabe auf stdout
print("Dies ist eine Nachricht auf stderr", file=sys.stderr) # Ausgabe auf stderr
print(f"{sys.argv} wartet zehn Sekunden:")
time.sleep(10)
print(f"{sys.argv} wird beendet.")
sys.exit(42)

# file subprocess_test.py
import subprocess, sys

if __name__ == "__main__":
    print(f"{sys.argv} startet einen Subprozess:")
    result = subprocess.run(["python", "write_stdout_stderr.py"],
        capture_output=True, text=True, check=False)
    print(result.stdout)
    print(result.stderr)
    print(f"Der returncode war: {result.returncode}")
```

A.5 Aufgabe: pinglawine.py

- (4) Schreibe das Programm `pinglawine.py`, das eine IP-Adresse mit Netzwerk-Suffix einliest und anschließend hintereinander alle gültigen Host-Adressen des gesamten Subnetzes anpingt.

Hinweis: Alle Pings werden der Reihe nach hintereinander ausgeführt – also blockierend ohne Multithreading: Erst wenn ein Fenster geschlossen wird, dann beginnt der nächste ping in einem neuen Fenster.

```
res = subprocess.run(['ping ', adresse], creationflags=subprocess.CREATE_NEW_CONSOLE)
```

B Remote-Befehle ausführen – Paramiko

Paramiko ist eine Python-Bibliothek, die es ermöglicht, SSH-Verbindungen herzustellen, Remote-Befehle auszuführen und Dateien zu übertragen. Siehe z.B. * <https://codilime.com/blog/python-paramiko-and-netmiko-for-automation/>

B.1 Paketinstallation und Grundprinzip von Paramiko

Dazu muss zunächst das Paket `paramiko` installiert werden (entweder im PyCharm Paketmanager oder in der Shell mit `bash pip install paramiko`).

Anschließend kann man sich folgendermaßen mit einem Remote-Server mittels Benutzernamen / Passwort authentifizieren und einen Befehl ausführen:

```
import paramiko
sshclient = paramiko.SSHClient() # Erstelle ein SSH-Client-Objekt
sshclient.set_missing_host_key_policy(paramiko.AutoAddPolicy())
sshclient.connect('hostname', username='username', password='password') # Verbindung herstellen
stdin, stdout, stderr = sshclient.exec_command('ls -l') # Remote-Befehl ausführen und Output holen
print(stdout.read().decode())
sshclient.close() # Verbindung schließen
```

B.2 Authentifizierung mit Schlüsselpaar

Im 2. Jahrgang haben wir in SYT/BS bzw. ITSI gelernt, wie man ein Schlüsselpaar (*public + private key pair*) zur sicheren Authentifizierung generiert und einsetzt. Wir machen nochmals rasch die Aufgaben aus der damaligen Übung:

- (5) Am Server wird der *public-key* eines Benutzers hinterlegt. Jede/r der den passenden *private-key* besitzt, darf sich einloggen.

- a. Starte eine Kali-Linux VM (mit Netzwerk = host only), die wir als Remote-Server verwenden.
 - b. Erzeuge auf dem Host-Rechner (= der Rechner von dem aus man sich auf den Remote-Server anmelden möchte: Windows⁵ oder Linux) mit `ssh-keygen` ein Schlüsselpaar (ohne *Passphrase*) für `junioradmin`.
 - Welche Dateien werden erzeugt – wie lautet deren absoluter Pfad?
 - Welche zusätzliche Informationen werden angezeigt?
 - c. Kopiere den public-key **vom Host in** die Linux-VM:
 - Je nachdem, welches Host-System verwendet wird, folgendes Kommando benutzen:
 - Windows-Host:


```
type $env:USERPROFILE\.ssh\id_rsa.pub | ssh user@IP-Adresse "cat >> .ssh/authorized_keys"
```

 Erkläre die Funktionsweise dieses Work-Arounds!
 - Linux-Host: `ssh-copy-id user@IP-Adresse`
 - Welche Daten werden kopiert (Quelle – Ziel)?
 - Warum darf der *private-key* die eigene Maschine **NIEMALS** verlassen?
 - d. Testen: Was passiert jetzt beim Verbinden mit `ssh user@IP-Adresse`? Wird nach einem Passwort gefragt?
 - e. Warum gilt die Authentifizierung mittels Schlüsselpaar als sicherer⁶ als ein Login mittels herkömmlicher Passwörter?
- (6) Nachdem die Anmeldung über das Schlüsselpaar funktioniert, können wir das auch mittels *Paramiko* aus einem Python-Programm verbinden: Prinzip:

```
def key_based_connect(host, passphrase=None):
    username = "kali"
    keyfile = r"C:\Users\harald\.ssh\id_rsa"
    if not passphrase:
        passphrase = getpass(f"Enter passphrase for key: {keyfile}")
    pkey = paramiko.RSAKey.from_private_key_file(keyfile, password=passphrase)
    client = paramiko.SSHClient()
    policy = paramiko.AutoAddPolicy()
    client.set_missing_host_key_policy(policy)
    client.connect(host, username=username, pkey=pkey, )
    return client
```

Hinweis: `getpass()` muss man importieren und funktioniert innerhalb der Python-IDE eventuell nicht ganz einwandfrei – daher ggf. in einer Python-Shell testen.

- (7) Aufgabe: schreibe das Programm `journalctlremote.py`, das eine Zeit in Minuten einliest. Danach verbindet es sich mittels Schlüsselpaar mit dem Server (IP-Adresse kann hardcoded sein) und holt alle Journal-Log-Einträge der letzten Minuten.
- (8) Netzwerkautomatisierung mit Netmiko

Paramiko ist ein generisches SSH-Modul, das sich besonders zur Kommunikation mit dem OpenSSH-Protokoll eignet. Die meisten Netzwerkgeräte benutzen proprietäre SSH-Implementierungen, die nicht 100% kompatibel zu OpenSSH sind. Zu diesem Zweck ist *Netmiko* die geeignetere Bibliothek, weil diese zusätzliche Funktionen bietet, die speziell für die Verwaltung von Netzwerkgeräten unterschiedlicher Hersteller entwickelt wurden.

Als abschließende Vorbereitung für weiterführende Netzwerktechnik-Übungen ist diese Seite (ab Abschnitt *Why isn't Paramiko enough for network automation?*) lesen: <https://codilime.com/blog/python-paramiko-and-netmiko-for-automation/>

⁵Ja, mittlerweile funktioniert das tatsächlich auch unter Windows.

⁶vorausgesetzt man passt gut auf den geheimen private-key auf!