Kurzeinführung in Skripting

Um immer wiederkehrende Aufgaben zu automatisieren, können sogenannte Skripte verwendet werden. Diese Skripte sind Textdateien, die z.B. Befehlsabfolgen enthalten, die normalerweise sequenziell auf der Kommandozeile eingegeben würden. Beim Ausführen werden diese Texte von einem Interpreter [1)] analysiert und die enthaltenen Befehle werden ausgeführt. Die einfachste Variante sind die `bash`-Skripte, da hier die Befehle wie auf der Kommandozeile eingegeben werden. Die Kommandozeile ist nämlich nichts anderes als das Programm `/bin/bash`.

Zusammenfassung der notwendigen Schritte:

1. Textdatei erstellen/speichern (z.B. `test.sh`)
2. in der ersten Zeile den Interpreter benennen (`#!/bin/bash`)
3. Datei als ausführbar markieren (`chmod +x DATEINAME`)

Textdatei erstellen

Eine Textdatei kann mit einem Texteditor der Wahl erstellt werden. Hier einige Beispiele für Texteditoren:

- gedit (grafisch)
- leafpad (grafisch)
- vi / vim (Kommandozeile)
- nano (Kommandozeile)
- bluefish (grafisch)
- emacs (grafisch/Kommandozeile)

Am Einfachsten erstellt man eine neue Datei, in dem den Editor öffnet und anschließend die Datei unter dem gewünschten Namen speichert. Alternativ kann man auch mit dem Befehl `touch DATEINAME` eine leere Datei generieren („berühren“) und anschließend mit dem Editor öffnen.

Interpreter benennen

Am Anfang eines Skriptes muss man Linux mitteilen, mit welchem Interpreter der folgende Text analysiert werden soll. Dies geschieht durch den speziellen Kommentar `#!/bin/bash` [2)]. Dabei wird nach dem Ausrufezeichen der Pfad zum Interpreter angegeben. Hier: `/bin/bash`.

test.sh

```
#!/bin/bash
echo "Das ist ein Test und gibt nur diesen Text aus."
echo "Jetzt wird der Ordnerinhalt angezeigt:"
ls -l
```

Textdatei ausführbar machen

Damit das Skript von Linux überhaupt ausgeführt werden kann, muss es zunächst als ausführbar gekennzeichnet werden (s.o.).

```
chmod +x test.sh
```

Dabei bedeutet `chmod` so viel wie `change modus` und `+x` setzt das e`X`ecute Bit. Danach wird die Datei als ausführbar gelistet, was unter Ubuntu standardmäßig mit einer grünen Schrift angezeigt wird. Ausführen lässt sich das Skript nun, indem man den vollständigen Pfad angibt. Wenn man sich im Ordner befindet, in dem das Skript gespeichert wurde, wäre dies:

```
./test.sh
```

Mit `./` ist der aktuelle Pfad gemeint [3]. Alternativ kann man ein Skript auch in einen Ordner des Default-Suchpfades [4] also z.B. `/home/USER/.bin/HIERHIN`

Die zweite Variante hängt aber vom jeweiligen System ab.

Tipps zum Aufbau eines Skripts

Skripte werden häufig im Bereich Administration von Servern und Netzelementen eingesetzt um Wiederkehrende Aufgaben zu erledigen. Hierzu zählen:

- Installationskripte z.B. für neue Software
- Konfiguration z.B. von IP-Adressen oder Routen
- Anlage von Usern in einem Active Directory
- ...

Die Liste lässt sich beliebig fortsetzen. Entscheidend für ein gutes Skript ist die gute Lesbarkeit. D.h. man sollte die einzelnen Abschnitt gut kommentieren, damit man später noch versteht, was das Skript eigentlich tut. Mit etwas Übung kann man Skripte auch so schreiben, dass sie wiederverwertbar sind.

Wie erreicht man nun ein gutes Ergebnis? Hierzu sollte man sich überlegen, welche Befehle benötigt werden. Diese kann man zunächst händisch auf der Kommandozeile ausprobieren und sie anschließend in seine Textdatei aufnehmen. Hat man alle benötigten Befehlszeilen zusammen, dann geht es an das Sortieren. Häufig ist die Reihenfolge wichtig (z.B. bei Firewall-Regeln oder beim Aktivieren einer Route). Hierbei ist es zweckmäßig zunächst einen definierten Zustand herzustellen.

Beispiel: Routen in die Routing-Tabelle bringen

Arbeitet man an einem Testsystem könnte man z.B. zunächst die eingesetzten Interfaces löschen[5]. Dadurch werden auch alle Routen, die bereits anderweitig auf diesen Interfaces konfiguriert wurden gelöscht. Diese könnten ansonsten eventuell stören. Danach beginnt man mit der Konfiguration der IP-Adressen, denn man kann erst Routen nur konfigurieren, wenn dem System die Netze bekannt sind. Anschließend fügt man die zusätzlichen Routen hinzu.

routing_sample.sh

```
#!/bin/bash
# alle IP-Adressen und Routen an eth1 löschen
sudo ip addr flush eth1

# IP-Adresse 10.0.0.20 auf Interface eth1 setzen
sudo ip addr add dev eth1 10.0.0.20/8

# Default-Route setzen (Standard-Gateway 10.0.0.1)
sudo ip route add default via 10.0.0.1
```

Beispiel: telnet-Verbindung zu einem Switch aufbauen

Mit dem folgenden Script wird eine Verbindung mit Hilfe von `nc`[6] zu einer IP-Adresse aufgebaut. Im Anschluss werden bis zum `EOF`[7] die einzelnen Befehle zeilenweise übertragen. Die folgende Tabelle zeigt die Befehle und ihre Wirkung auf dem Switch:

Befehl	**Wirkung auf Switch**
`schueler`	Benutzername wird als erstes vom Switch erwartet
`schueler`	Passwort für `schueler`
`system-view`	Wechsel in den Konfigurationsmodus
`display vlan`	Anzeigen der alten VLAN-Konfiguration
`undo vlan all`	Löschen der alten VLAN-Konfiguration
`y`	Bestätigung der Löschung
`vlan 2`	`VLAN-ID 2` anlegen durch Wechsel in den `vlan-2`-Bereich
`port Ethernet 1/0/1`	Port 1/0/1 in VLAN2 bringen
`display vlan 2`	neue Konfiguration von VLAN2 anzeigen
`vlan 3`	`VLAN-ID 3` anlegen durch Wechsel in den `vlan-3`-Bereich
`port Ethernet 1/0/2`	Port 1/0/2 in VLAN3 bringen
`quit`	Zurück auf höchste Konfigurationsebene (VLAN verlassen)
`interface Ethernet 1/0/5`	In Konfigurationsebene von Port 1/0/5 wechseln
`port link-type trunk`	Betriebsmodus für aktuellen Port 1/0/5 auf `trunk` ändern
`port trunk permit vlan 3`	Port 1/0/5 tagged zu VLAN3 hinzufügen
`port trunk permit vlan 2`	Port 1/0/5 tagged zu VLAN2 hinzufügen
`quit`	Zurück auf höchste Konfigurationsebene (Port verlassen)
`quit`	Verbindung zu Switch beenden

telsession.sh

```
#!/bin/bash
# netcat (kurz nc) ist ein Kommandozeilen-Tool, mit dem Verbindungen zu
entfernten Systemen aufgebaut werden können.
# Verwendung: nc <IP-Adresse> <PORT>
nc 192.168.33.61 23 <<'EOF'
schueler
```

```
schueler
system-view
display vlan
undo vlan all
y
vlan 2
port Ethernet 1/0/1
display vlan 2
vlan 3
port Ethernet 1/0/2
quit
interface Ethernet 1/0/5
port link-type trunk
port trunk permit vlan 3
port trunk permit vlan 2
quit
quit
EOF
```

Beispiel: Automatische Synchonisieren von Dateien mit Luckybackup

Mit dem folgenden Script lässt sich die Synchronisation unter Linux automatisieren. Voraussetzung dafür ist das automatische Einbinden von USB-Laufwerken. Ein paar Anpassung sind nötig:

1. Pfad zum USB-Stick anpassen (hier `/media/USERNAME/USB_STICK_NAME`
2. mit Befehl `touch /media/USERNAME/USB_STICK_NAME/stick.ready` Testdatei auf dem USB-Stick erzeugen
3. Profil-Name von Luckyback eintragen (Default ist: default)

Wie funktioniert das Ganze?

Das Script prüft, ob die Datei `/media/USERNAME/USB_STICK_NAME/stick.ready` (natürlich auf den richtigen Stick) existiert. Sollte sie nicht existieren, dann legt sich das Script für 1 Sek schlafen. Wenn die Datei da ist, also der Stick vom System automatisch eingebunden wurde, dann wird Luckybackup mit dem Default-Profile ausgeführt. Hier können auch andere Profile ausgewählt werden. Dazu muss `default` durch den entsprechenden Profilnamen ersetzt werden.

luckybackup.sh

```
#!/bin/bash
# 1. Pfad zum USB-Stick anpassen (hier /media/USERNAME/USB_STICK_NAME
# 2. mit 'touch  /media/USERNAME/USB_STICK_NAME/stick.ready'  Testdatei
auf dem USB-Stick erzeugen
# 3. Profil-Name von Luckyback eintragen (Default ist: default)
x=0
while [ "$x" -lt 100 -a ! -e /media/USERNAME/USB_STICK_NAME/stick.ready
]; do
        x=$((x+1))
        sleep .1
```

```
        echo "Laufwerk fehlt"
done
echo "Profile -default- wird synchronisiert!"
luckybackup --silent default
```

Python

Eine weitere sehr mächtige Variante Prozesse zu automatisieren, liegt in der Verwendung eine Script-Sprache wie `python` oder `perl`.

python_sample.py

```
#!/usr/bin/python
# coding: utf8

# Import für Call https://docs.python.org/2/library/subprocess.html
from subprocess import call

# Import für os.system https://docs.python.org/2/library/os.html
import os

call(["ls", "-l"])

os.system("ps aux | grep -i apache > output_file")


# User anlegen
def createUser(name,username,password):
    encPass = crypt.crypt(password,"22")
    return  os.system("useradd -p "+encPass+ " -s "+ "/bin/bash "+ "-d
"+ "/home/" + username+ " -m "+ " -c \""+ name+"\" " + username)

testVar = raw_input("Ask user for something.")
print (testVar)
```

ssh mit python

Es gibt mehrere Implementierungen für ssh in python. Das Module `paramiko` wird hier vorgestellt.

Installation:

```
python3 -m pip --proxy https://192.168.21.91:3128 install --user paramiko
```

Beispiel-Script:

ssh_paramiko_test.py

```
#
https://daanlenaerts.com/blog/2016/01/02/python-and-ssh-sending-command
s-over-ssh-using-paramiko/

import paramiko
USERNAME='schueler'
PASSWORD='schueler' #kritisch bei offenen Strukturen; hier besser keys
verwenden
HOST='192.168.33.80'

ssh = paramiko.client.SSHClient()
ssh.set_missing_host_key_policy(
 paramiko.AutoAddPolicy())

ssh.connect(HOST, username=USERNAME, password=PASSWORD)
stdin, stdout, stderr = ssh.exec_command('ip address print')
while not stdout.channel.exit_status_ready():
                # Print data when available
                if stdout.channel.recv_ready():
                    alldata = stdout.channel.recv(1024)
                    prevdata = b"1"
                    while prevdata:
                        prevdata = stdout.channel.recv(1024)
                        alldata += prevdata

                    print("1:"+str(alldata, "utf8"))

stdin, stdout, stderr = ssh.exec_command('ip address add
address=22.0.0.1/24 interface=ether4')
while not stdout.channel.exit_status_ready():
                # Print data when available
                if stdout.channel.recv_ready():
                    alldata = stdout.channel.recv(1024)
                    prevdata = b"1"
                    while prevdata:
                        prevdata = stdout.channel.recv(1024)
                        alldata += prevdata

                    print("2:"+str(alldata, "utf8"))
```

ssh_ssh_test.py

```
from ssh.session import Session
from ssh import options
# Username für Login
USERNAME= 'schueler'
# Zielmaschine hier mikrotik
HOST = '192.168.33.80'
```

```
# Aufbau der ssh-Session
s = Session()
s.options_set(options.HOST, HOST)
s.connect()

# Login mit USERNAME
s.userauth_agent(USERNAME)

chan = s.channel_new()
chan.open_session()
chan.request_exec('ip address print')
"""
# Ausgabe der Antwort
size, data = chan.read()
while size > 0:
    print(data.strip())
    size, data = chan.read()
"""
chan.request_exec('ip address add address=30.0.0.1/24
interface=ether3')
chan.request_exec('ip address print')

# Ausgabe der Antwort
size, data = chan.read()
while size > 0:
    print(data.strip())
    size, data = chan.read()

chan.close()
```

ssh über bash

ssh_test.sh

```
#!/bin/bash

user="schueler"
host="192.168.33.80"
ip_addr="40.0.0.1"

ssh $user@$host<<EOF
/ip address print
/ip address add address=${ip_addr}/24 comment="MyNetz" interface=ether5
/ip address print
EOF
```

1)

z.B. `bash` oder `perl`
2)

ist das Kommentarzeichen; alles dahinter wird vom Interpreter selbst ignoriert
3)

vgl. pwd
4)

vgl. PATH-Umgebungsvariable
5)

flush
6)

nc: netcat
7)

EOF: end of file