2.1 Skripte/Einführung

2.1.1 Speichern einer Problemlösung

Zu einer gegebenen IP-Adresse im lokalen Netzwerk soll die MAC-Adresse ausgegeben werden. Nach langem Ausprobieren findet man folgende Lösung:

Terminal
```
schueler@debian964:~$ ping -c1 -w1 10.92.0.1 &> /dev/null
schueler@debian964:~$ /sbin/arp -n | grep 10.92.0.1 | egrep -o '[0-9a-f:]17'
11:22:33:44:55:66
```

In dem Moment wünscht man sich einen Speicher, in den man diese Lösung einwirft, und aus dem man sie später (durch Mausklick, Tastendruck oder Eingabe eines Befehlswortes (z.B. `startbackup`) wieder herausholen kann.

Eine Möglichkeit ist das Shellskript: Eine Datei, die alle nötigen Befehlszeilen enthält und die man wie ein „echtes“ Programm später immer wieder abspielen kann.

Für manche Probleme reicht das vielleicht noch nicht aus, und man braucht Variablen, Schleifen und Verzweigungen, vielleicht auch Unterfunktionen. Auch dann kann ein Shellskript helfen.

Vielleicht möchte man dem Kunden für die Lösung eines kleinen Problems ein kurzes Programm (evtl. mit graphischer Oberfläche) anbieten. Selbst dann bietet sich ein Shellskript an.

2.1.2 Erstellen eines Shell-Skriptes

Die Befehle, die man sonst per Tastatur eingegeben hat, werden in eine Datei geschrieben (wie bei einer Batch-Datei unter einem anderen Betriebssystem):

Terminal
```
schueler@debian964:~$ nano sagwas.sh
```

Als Beispiel soll in `sagwas.sh` folgender Inhalt stehen:

sagwas.sh
```
1 pwd
2 uname -r
3 hostname
4 ps
5 SCHULE=CSBME
```

Man sieht: Für ein einfaches Skript braucht man keine besonderen Kenntnisse.

2.1.3 Ausführen eines Shell-Skriptes

Ein Shell-Skript kann nicht direkt auf der CPU laufen. Die CPU kennt nämlich keine Shell-Befehle. Stattdessen muss ein Shell-Skript immer durch eine Shell, z. B. die Bash, ausgeführt werden. Man sagt, die Shell *interpretiert* das Skript; die Shell ist ein *Interpreter* für die Shell-Sprache.

Und jetzt kommt es darauf an, *welcher* Shell-Prozess das Skript ausführen soll:

a) der aktuelle Shell-Prozess, der sowieso gerade läuft

b) eine neuer (zusätzlicher) Prozess, der durch die gerade laufende Shell gestartet werden muss

Diese beiden Möglichkeiten (Abbildung 1) werden jetzt vorgestellt.

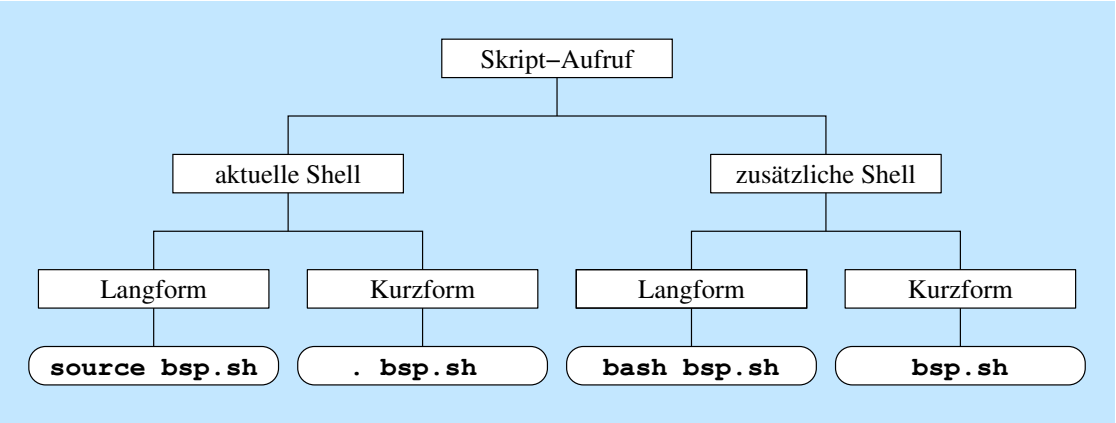

Abbildung 1: Möglichkeiten, ein Shell-Skript aufzurufen

2.1.3.1 Möglichkeit I: Ausführen im aktuellen Shell-Prozess Mit dem Befehl `source` kann man die in der Datei gespeicherten Befehlszeilen vom aktuellen Shell-Prozess ausführen lassen:

Terminal
```
schueler@debian964:~$ source sagwas.sh
/home/schueler
6.1.0-12-amd64
debian964
    PID TTY          TIME CMD
   7890 pts/1    00:00:00 bash
   9766 pts/1    00:00:00 ps
schueler@debian964:~$
```

Für den `source`-Befehl gibt es eine noch kürzere Schreibweise:

Terminal
```
schueler@debian964:~$ . sagwas.sh
```

Diese Methode (egal, ob `source` oder `cd`) hat allerdings Nachteile:

- Das Skript kann Umgebungsvariablen umsetzen
- Das Skript kann das aktuelle Verzeichnis verstellen
- Das Skript kann Shell-Einstellungen ändern

Alle diese Veränderungen sind nach dem Ende des Skriptes immer noch wirksam:

Terminal
```
schueler@debian964:~$ . sagwas.sh
...
schueler@debian964:~$ echo "$SCHULE"
CSBME
```

Deshalb benutzt man diese Arten des Aufrufs meistens für den Fall, dass ein Skript *nur* dazu dient, diese Einstellungen absichtlich herbeizuführen.

2.1.3.2 Möglichkeit II: Ausführen in einem neuen Shell-Prozess Will man einen neuen (zusätzlichen) Shell-Prozess aufrufen und in ihm das Skript ausführen, gibt man ein:

Terminal
```
schueler@debian964:~$ bash sagwas.sh
/home/schueler
6.1.0-12-amd64
    PID TTY          TIME CMD
   7890 pts/1    00:00:00 bash
   9827 pts/1    00:00:00 bash
   9828 pts/1    00:00:00 ps
debian964
schueler@debian964:~$ echo $SCHULE

schueler@debian964:~$
```

Man sieht: Nun laufen zwei Shell-Prozesse anstelle von einem. Außerdem: Mit dem Ende der Shell sind diesmal alle Veränderungen vergessen. Das ist ein Vorteil! Das Skript kann seine eigenen Einstellungen so viel verbiegen, wie es das will – nach dem Ende des Skripts ist alles wieder wie vorher.

Was ist hier passiert?

- Es wird ein neuer Shell-Prozess gestartet; als Parameter erhält er den Skript-Namen .
- Der neue Shell-Prozess öffnet das Skript und interpretiert es Zeile für Zeile.
- Nach Ende des Skriptes beendet sich der neue Shell-Prozess.

Auch hier gibt es eine Abkürzung: Oft möchte man ein Skript genauso aufrufen können wie ein Binärprogramm – sei es aus Bequemlichkeit oder um zu verstecken, dass man "nur" ein Skript geschrieben hat. Dann muss man zuerst das x-Bit für das Skript setzen:

Terminal
```
schueler@debian964:~$ chmod a+x sagwas.sh
```

Danach sollte es reichen, reinfach den Pfadnamen des Skripts einzugeben:

Terminal
```
schueler@debian964:~$ sagwas.sh
sagwas.sh: Kommando nicht gefunden.
```

Falls das wie hier misslingt, befindet sich das Verzeichnis mit dem Skript (z.B. das aktuelle Verzeichnis `./`) nicht in der Suchpfadliste `$PATH`.Dann muss es mit angegeben werden:

Terminal
```
schueler@debian964:~$ /home/schueler/sagwas.sh
...
```

Oder mit relativem Pfadnamen:

Terminal
```
schueler@debian964:~$ ./sagwas.sh
...
```

Das ist zu umständlich. Also bringt man entweder das Skript in ein Verzeichnis, das in der Suchpfadliste genannt wird oder man erweitert Suchpfadliste um das aktuelle Verzeichnis. Hier die erste Variante für ein bestimmtes Verzeichnis:

Terminal
```
schueler@debian964:~$ PATH=$PATH:/home/schueler
schueler@debian964:~$ sagwas.sh
...
```

Und hier die zweite Variante, die überall funktioniert, egal, wie das aktuelle Verzeichnis gerade heißt:

Terminal
```
schueler@debian964:~$ PATH=$PATH:.
schueler@debian964:~$ sagwas.sh
```

Man beachte den Punkt (Zeiger auf das aktuelle Verzeichnis) am Ende der Befehlszeile!

Der Benutzer `root` sollte aus Sicherheitsgründen **niemals** das aktuelle Verzeichnis (mit dem Namen `"."`) in seiner Suchpfadliste haben! Sonst könnte jemand auf die Idee kommen, ein Programm namens `sl` in das Verzeichnis `/tmp/` zu stellen und darauf zu warten, dass `root` auch einma lnach `/tmp/` wechselt und dort aus Versehen statt `ls` den Befehl `sl` eingibt. In anderen bekannten Betriebssystemen ist das ein größeres Problem: Dort kann man das aktuelle Verzeichnis gar nicht aus der Suchpfadliste herausnehmen ...

2.1.4 Fremde Interpreter und die Hash-Bang-Zeile

Es ist übrigens auch möglich, ein Skript automatisch mit einem *anderen* Shell-Programm als dem aktuell laufenden aufzurufen. Dazu ergänzt man das Skript:

Terminal
```
schueler@debian964:~$ cp sagwas.sh sagwas2.sh
```

sagwas2.sh
```
1 #!/bin/sh
2 pwd
3 uname -r
4 hostname
5 ps
6 SCHULE-CSBME
```

Danach kann man es wie gewohnt mit seinem Namen aufrufen:

Terminal
```
schueler@debian964:~$ sagwas2.sh
...
    PID TTY          TIME CMD
   7890 pts/1    00:00:00 bash
  10172 pts/1    00:00:00 sh
  10173 pts/1    00:00:00 ps
```

Was man hier in der vorletzten Zeile sieht: Es wurde die Shell `sh` benutzt, nicht wie sonst `bash`.

Der Trick liegt in der ersten Zeile: Das Zeichen "#" leitet normalerweise eine Kommentarzeile ein. Wenn aber in der ersten Skript-Zeile zu Beginn dieses Zeichen steht und direkt dahinter ein Ausrufezeichen folgt, dann wird der Rest der Zeile als Pfadname des Kommando-Interpreters aufgefasst (*hash-bang-* oder *shebang-Zeile*). Mit diesem Interpreter wird anschließend die Datei noch einmal geöffnet. Auf diese Art kann man auch Perl, Python, Tcl/Tk oder andere Skriptsprachen benutzen[1]:

hello.py
```
1 #!/usr/bin/python3
2 print("Hello, World!")
```

[1] Jedenfalls mit allen Programmiersprachen, in denen ein Hash einen Kommentar einleitet

Terminal
```
schueler@debian964:~$ chmod 755 hello.py
schueler@debian964:~$ PATH=$PATH:.
schueler@debian964:~$ hello.py
Hello, World!
```

Python-Skripte müssen übrigens nicht auf `.py` enden (und Shell-Skripte auch nicht auf `.sh`):

Terminal
```
schueler@debian964:~$ mv hello.py hello
schueler@debian964:~$ hello
Hello, World!
```

Egal wie ein Skript nun heißt: Im Allgemeinen ist es eine gute Konvention (=Brauch), die gewünschte Shell in der ersten Zeile anzugeben, auch dann, wenn es nur die normale Bash ist.

2.1.5 Einrichtung der Arbeitsumgebung mit Skripten

Sobald man sich im Textmodus einloggt, sind in der aktuellen Shell einige Voreinstellungen gesetzt. So hat die Eingabeaufforderung ein bestimmtes Aussehen, zeigt z. B. den aktuellen Benutzer und ein Dollarzeichen. Genauso ist es beim Öffnen eines Terminalfensters in der GUI.

Verantwortlich dafür sind bestimmte Shell-Skripte. Sie werden beim Start von der Shell mit dem Befehl `source` eingelesen.

2.1.5.1 Wo liegen die Skripte? Es gibt mehrere dieser Skripte. Das liegt daran, dass mehrere Leute Einstellung vornehmen können:

a) Der Systemverwalter hat die Möglichkeit, für alle Nutzer bindende Vorgaben einzustellen. Die Dateien für diese Einstellungen liegen im Verzeichnis `/etc`.

b) Der einzelne Nutzer kann seine eigenen Einstellungen vornehmen. Die Dateien dafür liegen als versteckte Dateien (Dateiname beginnt mit Punkt) in seinem persönlichen Verzeichnis.

Zudem gibt es drei Betriebsarten der Shell:

a) Die Login-Shell: Wenn man sich per ssh oder auf der Textkonsole eingeloggt hat, kommt man direkt auf eine Login-Shell[2].

   Durch die Option `-login` kann man eine Bash ausdrücklich als Login-Shell aurufen.

b) Die interaktive Shell: Wenn man auf der GUI ein Terminalfenster öffnet, kommt man ohne Login direkt auf eine Shell, in der man interaktiv seine Befehle eingeben kann. Genauso ist es nach Aufruf des Bash-Programms ohne Parameter, falls Eingabe und Ausgabe mit Tastatur und Bildschirm verbunden sind.

   Durch die Option `-i` kann man eine Bash ausdrücklich als interaktive Shell aurufen.

c) Die nicht-interaktive Shell: Wenn man ein Shell-Skript mit Nennung des Skriptnamens aufruft, wird eine nicht-interaktive Shell gestartet. Genauso ist es, wenn man die Shell mit dem Skriptnamen als Parameter aufruft.

Die einzelnen Betriebsarten benötigen unterschiedliche Einstellungen. Am einfachsten ist es mit der nicht-interaktiven Shell: Sie muss meistens nur die Einstellungen der aufrufenden Shell erben. Wenn man will, kann man ihr über den Variablennamen `$BASH_ENV` den Namen einer einzulesenden Datei mitgeben.

Ganz anders ist es bei der Login-Shell: Hier kann man nichts voraussetzen; alle relevanten Einstellungen müssen erst eingelesen werden.

Die Tabelle 1 zeigt, welche Dateien beim Start und beim Beenden der Bash eingelesen werden.

[2]Die `bash` erkennt die Betriebsart daran, dass sie mit dem Namen `-bash` aufgerufen wurde.

Betriebsart	Login-Shell	Interaktive Shell	Nicht-interaktive Shell

Start-Dateien	`/etc/profile`
	`~/.bash_profile`
	`~/.bash_login`
	`~/.profile`
		`/etc/bash.bashrc`
		`~/.bashrc`
			`$BASH_ENV`

Stop-Dateien	`~/.bash_logout`	—	—

Tabelle 1: Einstellungsdateien bei der Bash

2.1.5.2 Shell-Optionen für die Einrichtungs-Skripte In den oben genannten Dateien kann man die Einstellungen auf verschiedene Weise vornehmen:

a) Es gibt Umgebungsvariablen, die das Aussehen und Verhalten der Shell beeinflussen: `$PATH`, `$PS1` bis `$PS4` und viele weitere.

b) Mit dem Befehl `set` kann man allerhand einstellen: `set -f` schaltet die Wildcard-Expansion aus. Das Gegenteil erreicht man mit dem Pluszeichen: `set +f`

c) Weil bei set die Buchstaben nicht mehr reichten, hat man die Option `set -o` eingeführt, hinter der ein Wort steht: `set -o ignoreeof` schaltet die Möglichkeit ab, durch EOF bzw. Strg - D die Shell zu verlassen. Mit `set +o ignoreeof` macht man dies wieder rückgängig.

d) Für noch weitere Optionen gibt es `shopt`: Mit `shopt -s autocd` bewirkt man, dass man nur durch Nennung des Verzeichnisnamens (ohne `cd`) in ein Verzeichnis wechselt. Mit `shopt -u autocd` schaltet man dieses Verhalten wieder ab.

e) Für den Benutzer sind `alias`-Befehle nützlich. Mit ihnen kann man Abkürzungen für oft benutzte Befehle einbauen:

Terminal
```
schueler@debian964:~$ alias la="ls -la"
schueler@debian964:~$ alias clean="rm *~ *.BAK"
schueler@debian964:~$ alias hello="echo Hello, World!"
schueler@debian964:~$ hello
Hello, World!
```

2.1.6 Ergebnis

Die oben genannte Lösung packt man also in ein Skript:

```
1 #!/bin/bash
2 ping -c1 -w1 10.92.0.1 &> /dev/null
3 /sbin/arp -n | grep 10.92.0.1 | egrep -o '[0-9a-f:]{17}'
```

[title=mac1.sh] Man setzt die Berechtigungen und eventuell die Suchpfadliste:

Terminal
```
schueler@debian964:~$ chmod a+x mac1.sh
schueler@debian964:~$ PATH=$PATH:.
```

Und dann ruft man es auf:

Terminal
```
schueler@debian964:~$ mac1.sh
11:22:33:44:55:66
```