2.2 Skripte/Variablen

2.2.1 Gibt es in Shell-Skripten Variablen?

Beim Beispiel des letzten Themas war die IP-Adresse, nach der gesucht wurde, fest:

```
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] Es wäre schön, wenn man die Adresse irgendwie an das Skript übergeben könnte. Außerdem taucht sie mehrfach im Quelltext auf, was jede Änderung umständlich macht.

In anderen Programmiersprachen gibt es Variablen, in die man Werte speichern kann, um sie später wiederzuholen. Wie sieht das in einem Skript aus?

Das Mittel der Shell ist die *Variablen-Substitution*. Zur Erinnerung: Mit der folgenden Befehlszeile kann man eine Shell-Variable auf einen Wert setzen:

Terminal

```
schueler@debian964:~$ ORT="Bad Salzuflen"
```

Mit der Variablen-Substitution (eingeleitet durch das Dollar-Zeichen) kommt man an den Inhalt:

Terminal

```
schueler@debian964:~$ echo "$ORT"
Bad Salzuflen
```

2.2.2 Fakten zur Variablen-Substitution

Ein Variablenname darf aus Buchstaben, Ziffern und Unterstrich bestehen. Das erste Zeichen des Variablennamens darf allerdings keine Ziffer sein. Und es ist üblich, Variablennamen groß zu schreiben.

Beim Setzen des Inhalts (erste Zeile) muss der Name links stehen und der Inhalt rechts vom Gleichheitszeichen. Außerdem dürfen links und rechts vom Gleichheitszeichen keine Leerzeichen (oder Tabs) eingefügt werden.

Bei der Ausgabe (zweite Zeile) muss ein Dollar-Zeichen vor den Namen geschrieben werden. Das Dollar-Zeichen bedeutet, dass der Variablenname hier durch den Variableninhalt ersetzt werden soll.

Will man den Namen innerhalb von Strings verwenden, muss man ihn durch geschweifte Klammern schützen:

Terminal

```
schueler@debian964:~$ echo Pizza $xJahreszeiten
Pizza
schueler@debian964:~$ echo Pizza ${x}Jahreszeiten
Pizza 4Jahreszeiten
```

Durch den Befehl `unset` kann man eine Variable (und ihren Inhalt) löschen:

Terminal

```
schueler@debian964:~$ unset x
schueler@debian964:~$ echo $x

```

2.2.3 Maskierung

Als Inhalt einer Variablen kann man eine beliebige Zeichenkette verwenden. Aber Achtung:

a) Sonderzeichen (kleiner, größer, Pipe-Symbol, Semikolon, ...) müssen *maskiert* werden, damit sie nicht durch die Shell interpretiert werden.

b) Das Leerzeichen dient der Shell zum Trennen von Worten. Auch Leerzeichen müssen deshalb maskiert werden.

Zum Maskieren reicht es aus, die Zeichenkette in Anführungszeichen zu setzen:

Terminal
```
schueler@debian964:~$ MEINTXT="So ein Tag, so wunderschön wie heute."
schueler@debian964:~$ echo "$MEINTXT"
So ein Tag, so wunderschön wie heute.
```

Maskieren (=unwirksam machen von Sonderzeichen) kann wieder auf drei Arten passieren:

a) Backslash vor jedem Sonderzeichen: `So\ ein\ Tag`

b) Text in Hochkomma: `'So ein Tag'`

c) Text in Anführungszeichen: `"So ein Tag"`

Die dritte Methode hat die Besonderheit, dass das Dollarzeichen selbst nicht maskiert wird. Es empfiehlt sich, bei jeder Benutzung eines Variablen-Inhalts mit dem Dollarzeichen diese Maskierung zu verwenden:

Terminal
```
schueler@debian964:~$ VERZEICHNIS="Mein neues Verz"
schueler@debian964:~$ mkdir $VERZEICHNIS
schueler@debian964:~$ ls -l
drwxr-x--- 2 schueler schueler 4096 14. Apr 17:56 Mein
drwxr-x--- 2 schueler schueler 4096 14. Apr 17:56 neues
drwxr-x--- 2 schueler schueler 4096 14. Apr 17:56 Verz
schueler@debian964:~$ mkdir "$VERZEICHNIS"
schueler@debian964:~$ ls -l
drwxr-x--- 2 schueler schueler 4096 14. Apr 17:56 Mein neues Verz
```

2.2.4 Shell- und Umgebungsvariablen

Will man alle Shell-Variablen ansehen, kann man das mit dem Befehl `set` machen. Das Ergebnis ist meistens eine ziemlich lange Liste (enthält allerdings auch Shell-Funktionen, dazu später).

Allerdings werden nicht alle Variablen von der Shell an aufgerufene Programme weitervererbt, sondern nur ein genau festgelegter Teil von ihnen. Dieser exklusive Teil heißt *Umgebung*, die entsprechenden Variablen heißen *Umgebungsvariablen*.

Durch den Befehl `export ABC` wird die Shell-Variable `ABC` in diesen Club der Umgebungsvariablen aufgenommen; damit ist sie dann für aufgerufene Programme sichtbar.

Mit dem Befehl `env` werden nur die Umgebungsvariablen ausgegeben[1]. Alternativ kann man auch den Befehl `export -p` verwenden.

Um also dem Browser `Firefox` mitzuteilen, dass er den Proxy-Server `10.1.1.3` als Benutzer `xy` mit dem Passwort `123` benutzen soll, gibt man ein:

Terminal
```
schueler@debian964:~$ http_proxy=\
"http://xy:123@10.1.1.3:8080/"
schueler@debian964:~$ export http_proxy
schueler@debian964:~$ firefox
```

Man kann Setzen und Exportieren auch zusammenfassen:

[1] Unter Windows gibt es diese Unterscheidung nicht. Dort sind alle Variablen Umgebungsvariablen. Der Befehl zur Ausgabe aller Variablen lautet aber ebenfalls `set`.

Terminal
```
schueler@debian964:~$ export http_proxy=\
"http://xy:123@10.1.1.3:8080/"
schueler@debian964:~$ firefox
```

Will man die Umgebungsvariable nur für dieses eine Programm nutzen, kann man es noch kürzer ausdrücken:

Terminal
```
schueler@debian964:~$ http_proxy=\
"http://xy:123@10.1.1.3:8080/" firefox
```

2.2.5 Skript und Variablen

Wenn man ein Skript aufruft, wie es dann mit den Variablen? Man muss da unterscheiden:

a) Ein Skript, das in der aktuellen Shell gestartet wurde (Befehl `source` oder `.`), kann alle Shell-Variablen der aktuellen Shell benutzen und ändern.

b) Ein Skript, das in einer eigenen Shell gestartet wurde (Befehl `bash` oder *Skriptname*), erbt nur die Umgebungsvariablen von seiner aufrufenden Shell. Es kann sie verändern, aber nicht an die aufrufende Shell zurückgeben.

Der erste Fall ist einfach und selten, deshalb soll es ab jetzt nur noch um den zweiten Fall (eigene Shell) gehen.

2.2.6 Geerbte Umgebungsvariablen

Das Skript wurde also in einer eigenen Shell gestartet und erbt die Umgebungsvariablen der aufrufenden Shell. Auf wichtige Variablen wie `$USER`, `$HOME` und `$PATH` kann also auch vom Skript aus zugegriffen werden. Sie sind praktisch die „globalen Variablen“ der Skriptprogrammierung[2]:

zeigvariablen.sh
```
1 #!/bin/bash
2 echo Suchpfadliste.: $PATH
3 echo Persoenl.Verz.: $HOME
4 echo Loginname.....: $USER
```

Terminal
```
schueler@debian964:~$ zeigvariablen.sh
Suchpfadliste.: /usr/bin:/bin:/usr/local/games:/usr/games:.
Persoenl.Verz.: /home/schueler
Loginname.....: schueler
```

2.2.7 Skript-interne Variablen

Ebenso kann man im Skript auch eigene Variablen setzen und benutzen:

eigenevariablen.sh
```
1 #!/bin/bash
2 ORT=Bielefeld                          # setze Variable ORT
3 echo "Herzlich willkommen in $ORT." # benutze Variable ORT
```

Terminal
```
schueler@debian964:~$ eigenevariablen.sh
Herzlich willkommen in Bielefeld.
```

[2]Mit dem Unterschied, dass Änderung nicht an die aufrufende Shell zurückgegeben werden können

2.2.8 Variablen von der Tastatur einlesen

Mit dem Befehl `read` kann man dem Benutzer die Möglichkeit geben, einen Variableninhalt auf der Tastatur einzugeben.

einlesen.sh

```
1 #!/bin/bash
2 echo -n "Bitte Postleitzahl eingeben: "
3 read PLZ
4 echo -n "Bitte ORT eingeben: " # -n: ohne Zeilenumbruch
5 read ORT
6 echo "Adresszeile: "
7 echo "$PLZ $ORT"
```

Terminal

```
schueler@debian964:~$ einlesen.sh
Bitte Postleitzahl eingeben: 31737
Bitte ORT eingeben: Rinteln
Adresszeile:
31737 Rinteln
```

Komfortablere Skripten benutzen dazu häufig die Befehle `dialog`, `kdialog` oder `zenity` (ehemals `gdialog`). Sie erlauben ein GUI-artiges Aussehen von Shellskripten.

2.2.9 Skript-interne Variablen explizit vereinbaren

Mit dem Befehl `declare` (ehemals `typeset` genannt) kann man Variablen gezielt vereinbaren und mit einem Typ versehen:

Terminal

```
schueler@debian964:~$ declare -i x # Ganzzahl
schueler@debian964:~$ x=BLA
schueler@debian964:~$ echo $x
0
schueler@debian964:~$ x=4
schueler@debian964:~$ echo $x
4
```

Dabei bedeuten:

- Ohne Option: String (=Zeichenkette, default)
- `-i`: Zahl, diese Variable ist immer numerisch
- `-a`: Array
- `-A`: Assoziatives Array
- `-r`: außerdem konstant

2.2.10 Daten an ein Skript übergeben: Umgebungsvariablen

Jedes Skript kann wieder ein anderes Skript aufrufen. Im folgenden Beispiel ruft `eltern.sh` das Skript `kind.sh` auf. Sinnvollerweise passiert das wieder in der Form, dass eine neue Shell gestartet wird.

Falls man nun eine eigene Variable X auch in anderen aufgerufenen Programmen oder Skripten bekannt machen möchte, muss man sie mit dem `export`-Befehl in die Umgebung *exportieren*. Dadurch wird X von einer Shell-Variable zu einer Umgebungs-Variablen.

export.sh

```
#!/bin/bash
echo "setze PLZ...." # Hier gibt es PLZ noch nicht
PLZ=33604            # Ab jetzt ist PLZ eine Shell-Variable
echo "---------------"
echo "set:"
set | grep PLZ  # anzeigen, ob Shell-Var.
echo "env:"
env | grep PLZ  # anzeigen, ob Umgeb.-Var.
echo "---------------"
echo "exportiere PLZ...."
export PLZ       # Und ab jetzt ist PLZ eine Umgebungs-Var.
echo "---------------"
echo "set:"
set | grep PLZ  # anzeigen, ob Shell-Var.
echo "env:"
env | grep PLZ # anzeigen, ob Umgeb.-Var.
echo "---------------"
echo "unsette PLZ...."
unset PLZ       # Ab jetzt ist PLZ wieder weg
echo "---------------"
echo "set:"
set | grep PLZ # anzeigen, ob Shell-Var.
echo "env:"
env | grep PLZ # anzeigen, ob Umgeb.-Var.
echo "---------------"
```

Terminal

```
schueler@debian964:~$ export.sh
setze PLZ....
---------------
set:
PLZ=33604
env:
---------------
exportiere PLZ....
---------------
set:
PLZ=33604
env:
PLZ=33604
---------------
unsette PLZ....
---------------
set:
env:
---------------
```

Zur Erinnerung: Mit `set` kann man sich alle Shell-Variablen anzeigen lassen, mit `env` bekommt man nur die Umgebungs-Variablen angezeigt.

Immer wenn die Shell jetzt einen neuen Prozess aufruft, werden Variablen-Name und -Inhalt in den Umgebungs-Speicher (*environment*) des neuen Prozesses kopiert und stehen dort zur Verfügung.

Hier ein Beispiel, in dem die Variable `VARX` übergeben werden soll; zuerst ohne `export`:

eltern1.sh

```
#!/bin/bash
echo "Hier ist das Eltern-Skript 1"
VARX="123"
echo "Ich rufe das Kind-Skript auf"
echo "mit der Einstellung"
echo -n "Shell-Var..: "
set | grep VARX
echo -n "Umgeb.-Var.: "
env | grep VARX
echo -e "\n--------------"
kind.sh
```

kind.sh

```
#!/bin/bash
echo "Hier ist das Kind-Skript"
echo "Ich sehe die Einstellung:"
echo -n "Shell-Var..: "
set | grep VARX
echo -n "Umgeb.-Var.: "
env | grep VARX
echo -e "\n--------------"
```

Terminal

```
schueler@debian964:~$ eltern1.sh
Hier ist das Eltern-Skript 1
Ich rufe das Kind-Skript auf
mit der Einstellung
Shell-Var..: VARX=123
Umgeb.-Var.:
--------------
Hier ist das Kind-Skript
Ich sehe die Einstellung:
Shell-Var..: Umgeb.-Var.:
--------------
```

Hier bekam das Kind-Skript offenbar nichts von der Shell-Variablen `VARX` des Eltern-Skripts mit.

Und nun die Version mit `export`:

eltern2.sh

```
#!/bin/bash
echo "Hier ist das Eltern-Skript 2"
VARX="123"
export VARX
echo "Ich rufe das Kind-Skript auf"
echo "mit der Einstellung"
echo -n "Shell-Var..: "
set | grep VARX
echo -n "Umgeb.-Var.: "
env | grep VARX
echo -e "\n--------------"
kind.sh
```

Terminal
```
schueler@debian964:~$ eltern2.sh
Hier ist das Eltern-Skript 2
Ich rufe das Kind-Skript auf
mit der Einstellung
Shell-Var..: VARX=123
Umgeb.-Var.: VARX=123
--------------
Hier ist das Kind-Skript
Ich sehe die Einstellung:
Shell-Var..: VARX=123
Umgeb.-Var.: VARX=123
--------------
```

Hier wurde die Umgebungs-Variable `VARX` vom Eltern-Skript an das Kind-Skript vererbt.

Mit `export` kann man Variablen nur nach unten (vom Eltern- zum Kindprozess) vererben, niemals nach oben!

2.2.11 Berechnungen im Skript

Manchmal möchte man mit Shell-Variablen rechnen wie mit numerischen Variablen in normalen Programmiersprachen. Leider klappt das nicht so einfach:

Terminal
```
schueler@debian964:~$ X=3
schueler@debian964:~$ X=$X+1
schueler@debian964:~$ echo $X
3+1
```

Zur Abhilfe gab es früher das Programm `expr`, welches seine Parameter als Rechenaufgabe ansieht, die Aufgabe löst und das Ergebnis auf die Konsole ausgibt:

Terminal
```
schueler@debian964:~$ X=3
schueler@debian964:~$ expr $X + 1
4
```

Um damit eine Variable zu ändern, braucht man die Kommandosubstitution:

Terminal
```
schueler@debian964:~$ X=3
schueler@debian964:~$ X=$(expr $X + 1)
schueler@debian964:~$ echo "Ergebnis: $X"
Ergebnis: 4
```

Weil das umständlich und langsam ist (für jede Berechung muss `expr` aufgerufen werden), wurde die sogenannte *arithmetische Substitution* entwickelt.

2.2.12 Arithmetische Substitution

Die arithmetische Substitution unterscheidet sich von der Variablen-Substitution nur durch die Art der Klammern:

Terminal
```
schueler@debian964:~$ X=3
schueler@debian964:~$ echo $((X+1))# Arithmetische Substitution
4                                  # Ergebnis
```

Die alte (noch erlaubte) Schreibweise für die arithmetische Substitution sieht übrigens so aus:

Terminal
```
schueler@debian964:~$ echo $[X+1]
```

Wichtig ist:

- Das, was in den Klammern steht, wird ausgerechnet und durch das Ergebnis ersetzt (substituiert).
- Innerhalb der Klammern darf vor einem Variablenname ein Dollarzeichen stehen, muss aber nicht.
- Es dürfen beliebig viele Variablen, Konstanten und Operatoren innerhalb der Klammern stehen: Der Ausdruck `$((X+X+Y+Y+1))` ergibt $2 \cdot x + 2 \cdot y + 1$.
- Es gelten die von C/C++ und Java bekannten Vorrangregeln: Der Ausdruck `$((3*3+4*4))` ergibt $3^2 + 4^2 = 25$.
- Innerhalb der Klammern darf mit runden Klammern zusammengefasst werden: `$(((1+2)*(3+4)))` gibt 21; die Schreibweise `$[(1+2)*(3+4)]` ist dabei einfacher zu lesen.
- Es wird mit ganzen Zahlen gerechnet. Negative Zahlen sind möglich, Gleitkommazahlen dagegen nicht, auch nicht als Ergebnis: `$((38/10))` ergibt 3.
- Die aus C/C++ und Java bekannten arithmetischen Operatoren für ganze Zahlen sind erlaubt. Hinzu kommt der aus Python bekannte Operator `**` für Exponentiation: `$((2**7))` ergibt $2^7 = 128$.
- Man kann Variablen in einer arithmetischen Substitution neue Werte zuweisen, sowohl mit dem Gleichheitszeichen-Operator als auch mit den aus C/C++ und Java bekannten Abkürzungen: `$((X *= ++Y))` erhöht Y um eins und multipliziert X mit dem neuen Wert von Y. das Ergebnis wird der Shell-Variablen X zugewiesen.

2.2.13 Rechnen ohne Ausgeben: Stille Substitution

Was ist, wenn man durch die Shell einfach eine Variable erhöhen möchte, aber nichts auf der Konsole erscheinen soll?

Terminal
```
schueler@debian964:~$ X=4
schueler@debian964:~$ echo $((++X))
5                                  # Unerwuenschte Ausgabe.
schueler@debian964:~$ $((++X))
bash: 6: Kommando nicht gefunden.  # Aua.
```

Zu diesem Zweck gibt es die so genannte Stille Substitution, die einfach nur rechnet, das Ergebnis aber weder irgendwo einsetzt noch ausgibt:[3]

Terminal
```
schueler@debian964:~$ X=4
schueler@debian964:~$ ((++X))     # Erste Schreibweise
schueler@debian964:~$ let "++X"   # Zweite Schreibweise
```

[3] Die erste Schreibweise kommt daher, dass in neuen Bash-Versionen alles, was in doppelten Klammern steht, als arithmetischer Ausdruck angesehen wird (entsprechen in doppelten eckigen Klammern: als logischer Ausdruck).

2.2.14 Zusammenfassung

Man kann also tatsächlich in Skripten:

a) Variablen anlegen: `declare -i X`

b) Variablen setzen: `X=3`

c) Variablen vom Benutzer eingeben lassen: `read X`

d) Variablen berechnen: `((++X))`

e) Variablen ausgeben: `echo "$X"`

f) Variablen weitergeben: `export X`

Und so könnte das oben genannte MAC-Skript jetzt aussehen:

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

[title=mac2.sh] Der Aufruf ist dann:

Terminal

```
schueler@debian964:~$ IP=10.92.50.1 mac2.sh
11:22:33:44:55:66
```

Oder mit Benutzereingabe im Skript:

```
1 #!/bin/bash
2 read -p "Bitte IP eingeben: " IPADR
3 ping -c1 -w1 "$IPADR" &> /dev/null
4 /sbin/arp -n | grep "$IPADR" | egrep -o '[0-9a-f:]{17}'
```

[title=mac3.sh] Der Aufruf ist dann:

Terminal

```
schueler@debian964:~$ mac3.sh
Bitte IP eingeben: 10.92.50.1
11:22:33:44:55:66
```