2.6 Skripte/Schleifen

2.6.1 Mehrere Objekte bearbeiten

Das oben erstellte Programm `simplebackup6.sh` hat einen Nachteil: Man kann damit immer nur ein einziges Verzeichnis sichern. Was kann man tun, wenn man mit einer einzigen Befehlszeile mehrere Verzeichnisse sichern will?

Dazu haben die allermeisten Skript- und Programmiersprachen ein Mittel, die *Schleife*. Sie sorgt dafür, dass das Programm einen bestimmten Programmteil (den sogenannten *Schleifenrumpf*) eventuell mehrmals ausführt.

2.6.2 Arten von Schleifen

Es gibt mehrere Arten von Schleifen:

a) Die Endlosschleife (Abbildung 1, links: Struktogramm, Mitte: Programmflussdiagramm, rechts: Jackson-Diagramm) ist am einfachsten. Der Schleifenrumpf (=der betreffende Pro-

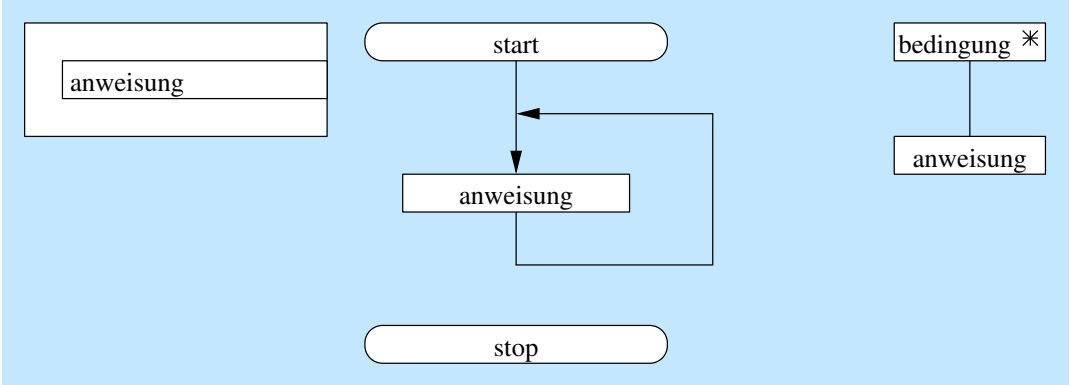

Abbildung 1: Endlosschleife

grammteil) wird ohne Ende immer wieder ausgeführt. Bei Systemen wie Ampelsteuerungen ist das durchaus sinnvoll.

b) Die kopfgesteuerte Schleife (Abbildung 2) hat ein Element mehr, nämlich eine Abfrage. Diese

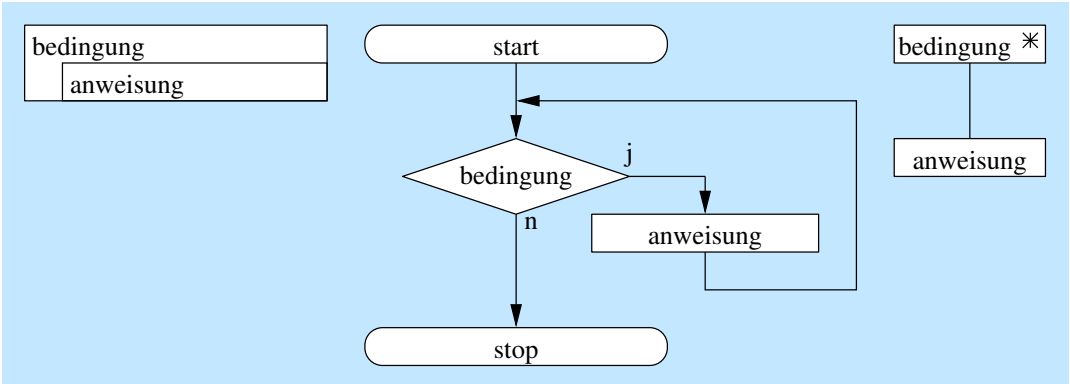

Abbildung 2: Kopfgesteuerte Schleife

Abfrage ähnelt der if-Anweisung bei einer Verzweigung.

Bei der kopfgesteuerten Schleife wird am Anfang der Schleife gefragt, ob eine bestimmte Bedingung erfüllt ist. Falls ja, dann wird der Schleifenrumpf ausgeführt. Falls nein, dann wird die Schleife verlassen. Und weil die Abfrage am Anfang steht, heißt sie auch *Schleifenkopf*.

Am Ende des Schleifenrumpfes passiert der Unterschied zur Verzweigung (=if-Abfrage): Hier springt der Programmablauf wieder zurück zur Abfrage, dem Schleifenkopf.

c) Die fußgesteuerte Schleife (Abbildung 3) fragt erst am Ende des Schleifenrumpfes, ob die Bedingung erfüllt ist.

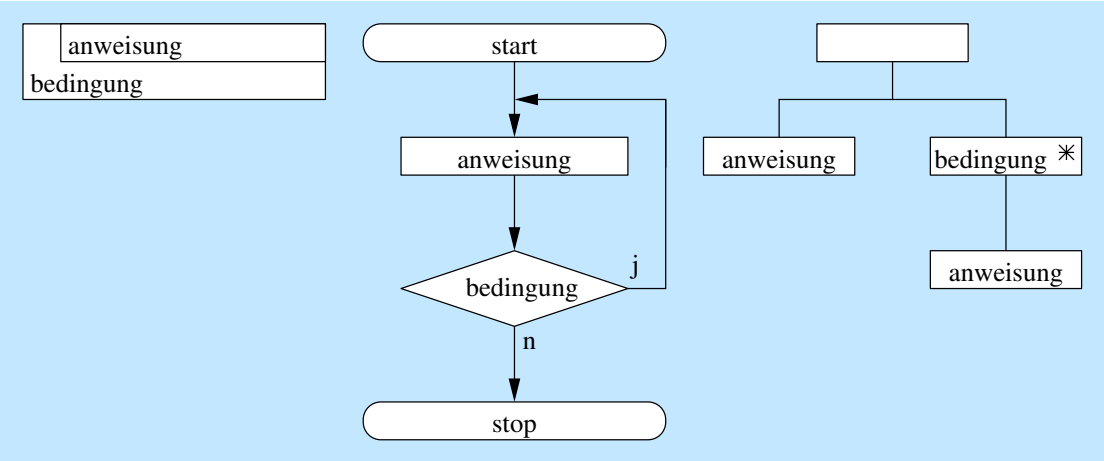

Abbildung 3: Fußgesteuerte Schleife

Deshalb wird der Schleifenrumpf bei dieser Schleifenform mindestens einmal ausgeführt.

d) Die Zählschleife (ohne Abbildung) ist eine Sonderform der kopfgesteuerten Schleife: Bei ihr legt der Programmierer schon zu Beginn fest, wie oft der Schleifenrumpf ausgeführt wird. Trotz dieser Einschränkung sind Zählschleifen sehr beliebt.

2.6.3 Die Kopfgesteuerte Schleife im Shell-Skript

Die kopfgesteuerte Schleife hat die Form:

```
while Kopf-Befehl
do
   Rumpf-Befehle
done
```

Die Logik ist genauso wie bei der Verzweigung:

a) Zu Beginn wird der Kopf-Befehl ausgeführt.

b) Falls der Kopf-Befehl erfolgreich war, also den Rückgabewert 0 ergab, werden die Rumpf-Befehle einmalig ausgeführt. Anders als bei der Verzweigung wird dann wieder zum Kopf gesprungen.

c) Falls der Kopf-Befehl nicht erfolgreich war, also einen Rückgabewert ungleich 0 ergab, springt der Programmablauf sofort zum Ende der Schleife.

Ein kleines Beispiel zeigt dieses Skript:

scpwarten.sh

```
#!/bin/bash
ZIELRECHNER=1.2.3.4
while ! ping -c1 -w1 "$ZIELRECHNER"
do
    echo "Warte auf Verbindung zu $ZIELRECHNER ..."
    sleep 20
done
echo "Verbindung vorhanden, kopiere Backup"
scp backup.zip "$ZIELRECHNER"
echo "fertig"
```

Ein Aufruf:

Terminal

```
schueler@debian964:~$ scpwarten.sh
Warte auf Verbindung zu 1.2.3.4
Warte auf Verbindung zu 1.2.3.4
... (einige Zeit danach) ...
Verbindung vorhanden, kopiere Backup
fertig
```

Der `ping`-Befehl gibt 0 zurück, falls der Zielrechner erreichbar ist. Durch das Ausrufezeichen davor wird diese Logik negiert: Die Befehlszeile `! ping ...` gibt 0 zurück, falls er nicht erreichbar ist. Die Konsequenz ist: Es wird immer wieder 20 Sekunden lang gewartet, solange der Zielrechner nicht erreichbar ist.

Wer das Ausrufezeichen (=Negation) nicht mag, kann den Schleifenkopf auch anders schreiben:

```
until ping -c1 -w1 "$ZIELRECHNER"
do
    ...
done
```

Beim Schlüsselwort `until` wird der Schleifenrumpf dann ausgeführt, wenn der Kopf-Befehl *nicht* erfolgreich war. Ansonsten sind `while` und `until` gleich[1].

2.6.4 Bedingungen testen mit `while` und `test`

Wie bei der Verzweigung kann man auch hier wieder den Befehl `test` verwenden (als Wort `test` oder als Paar eckiger Klammern `[]`). Damit kann man auch hier Zahlen und Strings vergleichen:

```
#!/bin/bash
NUTZER="$(who|wc -l)"
while [ "$NUTZER" -gt 3 ]  # <- Da ist er, der test-Befehl
do
    Es sind noch zu viele Nutzer da, naemlich "$NUTZER"
    NUTZER="$(who|wc -l)"
    sleep 1
done
echo "Jetzt kann es weitergehen ..."
```

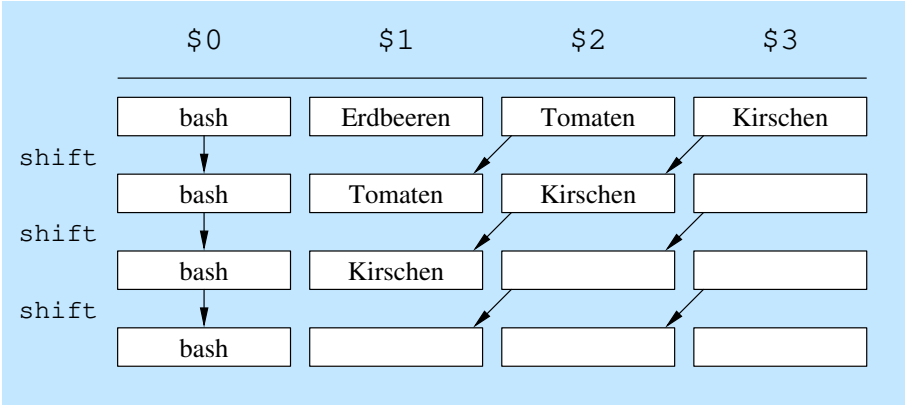

Abbildung 4: Wirkung des `shift`-Befehls

2.6.5 Parameter verarbeiten mit `while` und `shift`

Der in der Shell eingebaute Befehl `shift` (gibt es in Wind. genauso, heißt dort gleich) sorgt dafür, dass alle Parameter ab `$1` um eine Stelle nach links geschoben werden (Abbildung 4): `$3` wird zu `$2`, `$2` wird zu `$1`, und `$1` wird gelöscht. Mit diesem `shift`-Befehl kann man in der `while`-Schleife ganz einfach alle Parameter abarbeiten:

Terminal

```
schueler@debian964:~$ set Erdbeeren Tomaten Kirschen #setze $1,$2,$3
schueler@debian964:~$ while [ "$1" != "" ]
> do
>    echo "$1 sind rot."
>    shift
> done
Erdbeeren sind rot.
Tomaten sind rot.
Kirschen sind rot.
```

Der Schleifenrumpf bearbeitet immer nur `$1`. Aber der `shift`-Befehl sorgt dafür, dass jeder Parameter einmal zu `$1` wird.

2.6.6 Datei zeilenweise lesen mit `while` und `read`

Oft besteht der Wunsch, eine Datei *zeilenweise* zu verarbeiten: Für jede Zeile einer Datei sollen ein oder mehrere Befehlszeilen ausgeführt werden. Dazu nutzt man zwei Tricks:

- Der eingebaute Befehl `read` liest immer eine Eingabezeile (normalerweise von der Tastatur)
- Bei der Ein- und Ausgabeumleitung und der Befehlsverkettung gilt eine Schleife als **ein einziger** Befehl. Das heißt, die Ein- oder Ausgabedatei wird nur einmal geöffnet und geschlossen. Alles dazwischen betrifft die offene Datei

Und jetzt die Lösung:

Terminal

```
schueler@debian964:~$ while read ZEILE
> do
>   echo "$ZEILE" | cut -d"#" -f1
> done < /etc/protocols
```

[1] Vorsicht Delphi-Programmierer: while-do-done und until-do-done sind beides kopfgesteuerte Schleifen! Dagegen begrenzt repeat-until bei Delphi eine fußgesteuerte Schleife.

```
ip          0         IP
hopopt          0          HOPOPT
icmp          1          ICMP
igmp          2          IGMP
ggp          3          GGP
...
```

Anstelle der Eingabeumleitung kann man auch die Befehlsverkettung nehmen, um eine Reihe von Zeilen an eine Schleife weiterzugeben:

```
1 #!/bin/bash
2 /sbin/arp -n | while read ZEILE
3 do
4     echo -n "$ZEILE" | egrep -o [0-9a-f:]{13},"
5 done
```

2.6.7 Parameter abarbeiten mit der Listen-Zählschleife

Die einfachste Zählschleife hat die Form:

```
for Variablen-Name
do
   Rumpf-Befehle
done
```

Der Variablenname wird nacheinander auf die Werte der Positionsparameter `$1`, `$2`, `$3`, ... gesetzt. Anschließend jedes Mal werden die Rumpf-Befehle ausgeführt:

Terminal

```
schueler@debian964:~$ set Freitag Samstag Sonntag
schueler@debian964:~$ for TAG
> do
>    echo "$TAG mag ich."
> done
Freitag mag ich.
Samstag mag ich.
Sonntag mag ich.
```

Das Skript `calmonat.sh` soll so aussehen:

```
1 #!/bin/bash
2 for MONAT do
3     cal "$MONAT" 2030
4 done
```

Dann wird durch den Aufruf `calmonat.sh 1 2 3` der Kalender für Januar, Februar und März 2030 ausgegeben.

2.6.8 Wortlisten verarbeiten mit der Listen-Zählschleife

Die zweite (häufigste) Form der Zählschleife lautet:

```
for Variablen-Name in WortA WortB WortC
do
   Rumpf-Befehle
done
```

Hier wird der Variablenname nacheinander auf alle Werte in der Wortliste gesetzt. Beispiele:

```
#!/bin/bash
for KOLLEGE in willi herbert egon franz hans
do
    mail -s "WICHTIG!!!KEIN SPAM!! " "$KOLLEGE" < spamspamspam.txt
done
```

Die Wortliste kann auch durch eine Expansion oder Substitution entstehen:

```
#!/bin/bash
for DATEI in *.doc      # Wildcard-Expansion von Pfadnamen
do
   mv "${DATEI}" "${DATEI}_uralt_2023"
done
```

Mit dem Kommando `seq` kann man eine Wortliste erzeugen, die Zahlen enthält. Folgende Konstruktion gibt einen `ping`-Befehl an alle Clients im lokalen Netz:

```
#!/bin/bash
for i in $(seq 1 254)
do
   ping "172.20.1.$i"
done
```

Statt des `seq`-Befehls kann man auch die Klammer-Expansion verwenden:

```
#!/bin/bash
for i in {1..254}
do
   ping "172.20.1.$i"
done
```

2.6.9 Zählen mit der arithmetischen Zählschleife

Die dritte (neuere) Form der Zählschleife ist C++-ähnlich:

```
for ((AA1;AA2;AA3))
do
   Rumpf-Befehle
done
```

Der arithmetische Ausdruck AA1 wird einmal zu Beginn ausgeführt. Solange AA2 ungleich null ist, werden zuerst die Rumpf-Befehle ausgeführt und anschließend der Ausdruck AA3.

Terminal

```
schueler@debian964:~$ for ((i=0;i<4;++i))
> do
>    echo "Es ist $((i+12)) Uhr."
> done
Es ist 12 Uhr.
Es ist 13 Uhr.
Es ist 14 Uhr.
Es ist 15 Uhr.
```

2.6.10 Menüs anzeigen mit der interaktiven Zählschleife

Eine weitere Form der Zählschleife ist die *interaktive Zählschleife*. Mit ihr kann man ohne Aufwand ein kleines Text-Menü auf die Konsole zaubern:

```
select Variablen-Name in WortA WortB WortC
do
   Rumpf-Befehle
done
```

Hier passiert das Gleiche wie bei der entsprechenden Form der `for`-Schleife, allerdings wird zuerst die Wortliste (mit Nummern) auf den Bildschirm ausgegeben, anschließend kann der (Konsolen-) Benutzer eine gewünschte Position der Wortliste per Nummer auf der Tastatur eingeben und mit diesem Wort die Rumpf-Befehle ausführen (Ende mit Strg - D). Hier ein Beispiel:

Terminal

```
schueler@debian964:~$ select DATEI in muell*.txt
> do
>     rm "$DATEI" # Maskierung ist wichtig
>     echo "$DATEI entfernt"
> done
1) muellA.txt
2) muellB.txt
3) muellC.txt
#? 1
muellA.txt entfernt
#? 2
muellB.txt entfernt
#?  Strg  D
schueler@debian964:~$
```

Mit ↩ kann der Benutzer sich das Menü jederzeit wieder anzeigen lassen. Die Eingabeaufforderung (hier: `#?`) kann mit der Variablen `$PS3` angepasst werden:

Terminal

```
schueler@debian964:~$ PS3="Eingabe, Ende mit Strg-D: "
schueler@debian964:~$ select DATEI in muell*.txt
> do
>     rm "$DATEI" # Maskierung ist wichtig
>     echo "$DATEI entfernt"
> done
1) muellA.txt
2) muellB.txt
3) muellC.txt
Eingabe, Ende mit Strg-D: 1
muellA.txt entfernt
Eingabe, Ende mit Strg-D: 2
muellB.txt entfernt
Eingabe, Ende mit Strg-D:  Strg  D
schueler@debian964:~$
```

2.6.11 Endlosschleife

Für die Endlosschleife gibt es in der Shell keine eigene Konstruktion. Stattdessen nutzt man eine kopfgesteuerte Schleife mit einer Bedingung, die immer wahr ist. Dafür gibt es den Befehl `true`. Dieser Befehl tut nichts, gibt aber immer 0 zurück:

```
#!/bin/bash
while true
do
    echo hallo
    sleep 1
done
```

Oder man nimmt den Befehl `false`: Der tut ebenfalls nichts, gibt aber immer 1 zurück:

```
#!/bin/bash
until false
do
    echo hallo
    sleep 1
done
```

2.6.12 Fußgesteuerte Schleife

Die fußgesteuerte Schleife braucht man eher selten. In der Sprache der Shell-Skripte ist sie deshalb nicht realisiert. Durch geeignete Programmierung kann man sie umgehen.

2.6.13 Sprünge:`continue` und `break`

Wie in den Sprachen C und Java gibt es auch hier zwei Kontrollmöglichkeiten innerhalb der Schleifen:

- Innerhalb der Schleife kann durch den Befehl `continue` ein Rücksprung zum Anfang erreicht werden. Durch einen zusätzlichen Parameter kann bei verschachtelten Schleifen angegeben werden, zum Anfang welcher Schleife gesprungen wird (default ist 1, das meint die aktuelle innerste Schleife).
- Durch den Befehl `break` wird ein Verlassen der Schleife erreicht. Hier kann ebenfalls durch einen zusätzlichen Parameter bei verschachtelten Schleifen die Anzahl der zu verlassenden Schleifen genannt werden (default ist 1, dabei wird nur die innerste Schleife verlassen).

Beide Möglichkeiten können zu schwer zu findenden Programmfehlern führen und sollten deshalb in der Praxis nur sparsam verwendet werden.