Zusammenfassung Tag 21

Shell-Skripting

- Du kannst mehrere Befehle gebündelt in einer Kommandozeile ausführen, indem du sie per Semikolon abtrennst:

```
~$ Befehl; Befehl; Befehl
```

- Man kann Bash-Befehle auch gebündelt in einer Datei abspeichern, um sogenannte **Shell-Skripte** zu erstellen, die kompakt komplexe Logik ausdrücken (z.B. Backup einer Datenbank erstellen und auf einen FTP-Server laden) und direkt in der Shell ausgeführt werden
- Üblicherweise versieht man Shell-Skripte mit der **Endung .sh**, diese ist aber prinzipiell nicht notwendig
- Mit der **Shebang**-Zeile gibst du an, mit welchem Programm dein Skript ausgeführt werden soll; bei einem Bash-Skript schreibst du dazu in die erste Zeile:

```
#!/bin/bash
```

- Ohne explizite Shebang-Zeile werden Programme standardmäßig als Bash-Skripte ausgeführt
- Mit der Shebang-Zeile kannst du auch Skripte in anderen Programmiersprachen universell ausführbar machen; für Python-Programme benutzt du etwa als Shebang-Zeile (insbesondere kannst du dann auch auf die Dateiendung .py verzichten):

```
#!/bin/python3
```

- Wenn du Skripte per `./Skriptname` oder `bash Skriptname` ausführst, wird eine neue Bash-Shell gestartet, in der das Skript ausgeführt wird, Variablen aus dem Skript sind dann **nicht** im Rahmen der aktuellen Shell angelegt; wenn du hingegen ein Skript via `. Skriptname` innerhalb der aktuellen Shell ausführst, dann kannst du im Terminal auch die Variablen daraus abrufen
- Um bei der Ausführung eines Skripts **Parameter** zu übergeben, schreibst du diese einfach hinter den Skriptnamen:

```
./Skriptname.sh Parameter1 Parameter2 Parameter3
```

- Damit bei einer Ausgabe der Inhalt einer Variablen eingesetzt werden, muss der Variablennamen innerhalb **doppelter Anführungszeichen** stehen, bei Variablen in **einfachen Anführungszeichen** wird nur der Name eingesetzt, ohne jegliches Anführungszeichen wird der Inhalt aufgelöst (also z.B. die Tilde ~ als Pfad des Homeverzeichnisses)
- Als nützliches Pattern empfiehlt es sich, **lokale Variablen**, die du nur im Shell-Skript verwendest, klein zu schreiben, um zu verhindern, dass du versehentlich **globale Variablen**

(die groß geschrieben werden) abänderst; überdies kannst du Variablen als **readonly** markieren

- Du kannst mit dem Raute/Hash-Zeichen **#** ganze Zeilen in einem Bash-Skript **auskommentieren** (die Zeilen werden bei der Ausführung nicht beachtet; gilt natürlich nicht für die Shebang-Zeile)
- Mittels **Befehlssubstitution** kannst du die Ausgabe eines Befehls als Wert einer Variablen setzen:

```
Variablenname="$(Befehl)"
```

- Mit **if-Anweisungen** kannst du Logik-Abfragen implementieren:
  - du beginnst eine if-Anweisung mit **if** und beendest sie mit **fi**
  - Teil einer if-Anweisung ist die von der Shell mitgelieferte **Testfunktion**, die per geöffneter eckiger Klammer **[** aufgerufen wird und logische Ausdrücke (Vergleiche wie z.B. =, !=, <,…) auswertet
  - du musst zwingend **Leerzeichen** zwischen den eckigen Klammern und der enthaltenen Bedingung setzen, da jene als Parameter beim Aufruf der Testfunktion [übergeben wird
  - auch im logischen Ausdruck musst du auf **Leerzeichen** zwischen Variable, Operation und Wert achten
  - indem du den Code mit dem Tabulator **Tab** einrückst, stellst du ihn übersichtlicher dar:

```
if [ "$Variablenname"  Vergleichsoperation  Wert ]
then
   Befehle
fi
```

- Du kannst auch mehrere Vergleiche miteinander verbinden, d.h. zwei Shell-Befehle miteinander verketten
  - **and – Operator** (liefert 0, wenn beide Bedingungen erfüllt sind, sonst 1):

`[ Bedingung1 ] && [ Bedingung2 ]`	`[ Bedingung1 -a Bedingung2 ]`

  - **or – Operator** (liefert 0, sobald mindestens eine der Bedingungen erfüllt ist, sonst 1):

`[ Bedingung1 ] || [ Bedingung2 ]`	`[ Bedingung1 -o Bedingung2 ]`

- if-Anweisungen können mithilfe von **else** und **elif** erweitert werden, um auch die Fälle abzufangen, in denen die if-Bedingung nicht erfüllt ist, und darauf mit alternativen Befehlen zu reagieren

```
if [ "$Variablenname"  Vergleichsoperation  Wert ]
then
```

```
    Befehle
elif [ "$Variablenname"  Vergleichsoperation  Wert ]
    Befehle
else
    Befehle
fi
```

- Mittels Schleifen kannst du einen Block an Befehlen wiederholt ausführen lassen (in einem Bash-Skript wie auch im Terminal):
    - Bei der **for-Schleife** wird die Anzahl der Schleifendurchläufe durch die Angabe eines Zahlenbereichs (`{ErsteZahl..LetzteZahl}`) im Vorfeld festgelegt:

```
for Schleifenvariable in Zahlenbereich
do
    Befehle
done
```

    - Bei der **while-Schleife** hingegen wird mit einem Test (wie bei einer if-Anweisung) nur eine Abbruchbedingung festgelegt:

```
while [ Bedingung ]
do
    Befehle
done
```

- Daher besteht bei while-Schleifen die Gefahr von Endlosschleifen, also Schleifen, die nicht mehr verlassen werden und das Programm an dieser Stelle „einfrieren“:

```
while [ 1 ]
do
    Befehle
done
```

- Im Allgemeinen ist deshalb eine for-Schleife einer while-Schleife vorzuziehen