

Modul 122 Bash Scripts

M. Zollinger

Luca Mock
19.04.2023

Einführung	3
Git Repository	3
Aufgabe 1 Zahlenraten	4
Bash Code Zahlenraten	4
Zahlenraten mit Rate Versuchen	7
Zusatz	10
Aufgabe 2 Achtung Umleitung	12
ls -l	12
ls -l > ls-l.txt	12
rmdir	13
1 Fehlermeldung	13
2 Fehlermeldung, log file wird erstellt	13
3 Fehlermeldung mit Logfile Erstellung	14
ls -l grep ".txt"	15
ls -l sed -e "s/[aeio]/X/g"	18
Cat	18
1 Fehlermeldung	19
2 Fehlermeldung mit Logfile Erstellung	19
3 Fehlermeldung im Blackhole	20
Aufgabe 3 Du Bytes	21
--apparent-size:	21
--block-size=1:	22
Datei auflistung	22
Summe ausgeben	23
Aufgabe mit exec	23
Script duBytesSuender.sh	25
Script duBytesSuender.sh Erweiterung	26
Script "duBytesFileSizeDecreasing" Zusatz	27
Aufgabe 4 Text processing	32
Anwenden von cat, uniq und sort	32
Mehrere Dateien mit cat ausgeben	35
Eine Datei mit Zeilennummern versehen	36
Statt Seitenweise Man-Page Nur Das Wichtigste	37
Aufgabe 5 Umgestalten	38
Schritt 1 „Prüfen von Input und Output Parameter“	38
Schritt 2 „Alle deutschsprachigen Texte herausfiltern“	39
Schritt 3 Alle Zeilen mit einer „26“ herausfiltern	40
Schritt 4 „Spalte 1 und 2 sollen verschwinden“	42
Schritt 5 „Nur die Hauptgruppen ausgeben“	43

Aufgabe 6 Funktionen	45
Zusammenzählen	45
Zusatz "Rechner"	46
Befehlsübersicht	48

Einführung

In diesem Dokument werden die Aufgaben, die von Herrn Zollinger abgegeben wurden, gelöst und aufgezeigt. Die Aufgaben dienen dazu, ein besseres Verständnis für das Schreiben von Scripts und allgemein für Terminals zu erhalten. Außerdem kann man durch die entsprechenden Aufgaben viele Befehle, welche hier beschrieben und dokumentiert sind, lernen. Die einzelnen Aufgaben sind über die Übertitel dokumentiert. Am Ende des Dokuments findet man zudem noch eine Befehlsübersicht.

Die ganzen Scripts werden auf einem Mac im "Iterm2" Terminal ausgeführt. Außerdem verwende ich Fig, was einige Konfigurationen für Terminals vereinfacht und Autocomplete anbietet.

Die Scripts schreibe ich mit dem Editor Visual Studio Code und als Datei verwaltungssystem verwende ich Github

Für einige wenige Zusatzaufgaben ist es eventuell noch benötigt die coreutils zu installieren Dies macht man am besten mit Homebrew.

- Fig: <https://fig.io/>
- Github: <https://github.com/>
- Iterm 2: <https://iterm2.com/>
- Homebrew: https://brew.sh/index_de
- Visual Studio Code: <https://code.visualstudio.com/>
- Coreutils: <https://formulae.brew.sh/formula/coreutils>

Git Repository

https://github.com/lcmoc/M122_bash_scripts

Aufgabe 1 Zahlenraten

Bash Code Zahlenraten

Github:

- Script:

https://github.com/lcmoc/M122_bash_scripts/blob/0f5835daf046ae492496b146399fee02639e673e/Zahlenraten/zahlenraten.sh

- Diagram:

https://github.com/lcmoc/M122_bash_scripts/blob/main/Zahlenraten/bashScript.drawio

Script:

```
1 year ago | Author (10a)
#!/bin/bash

number=$((RANDOM % 10 + 1))

echo "Guess a number between 1 and 10:"

while true; do
    read guess

    if ! [[ "$guess" =~ ^[0-9]+$ ]] || [ "$guess" -gt 10 ]; then
        echo "Please enter a valid number between 1 and 10"
        continue
    fi

    if [ "$guess" -lt "$number" ]; then
        echo "Too low. Guess again:"
        continue
    fi

    if [ "$guess" -gt "$number" ]; then
        echo "Too high. Guess again:"
        continue
    fi

    echo "Congratulations! You guessed the number."
    break
done
```

Ausgabe:

```
apple ➤ ~/Documents/schule/3Lehrjahr/M122 ➤ ⚡ main !1 ?5 ➤ sh ./Zahlenraten/zahlenraten.sh
Guess a number between 1 and 10:
5
Too high. Guess again:
2
Too high. Guess again:
1
Congratulations! You guessed the number.
```

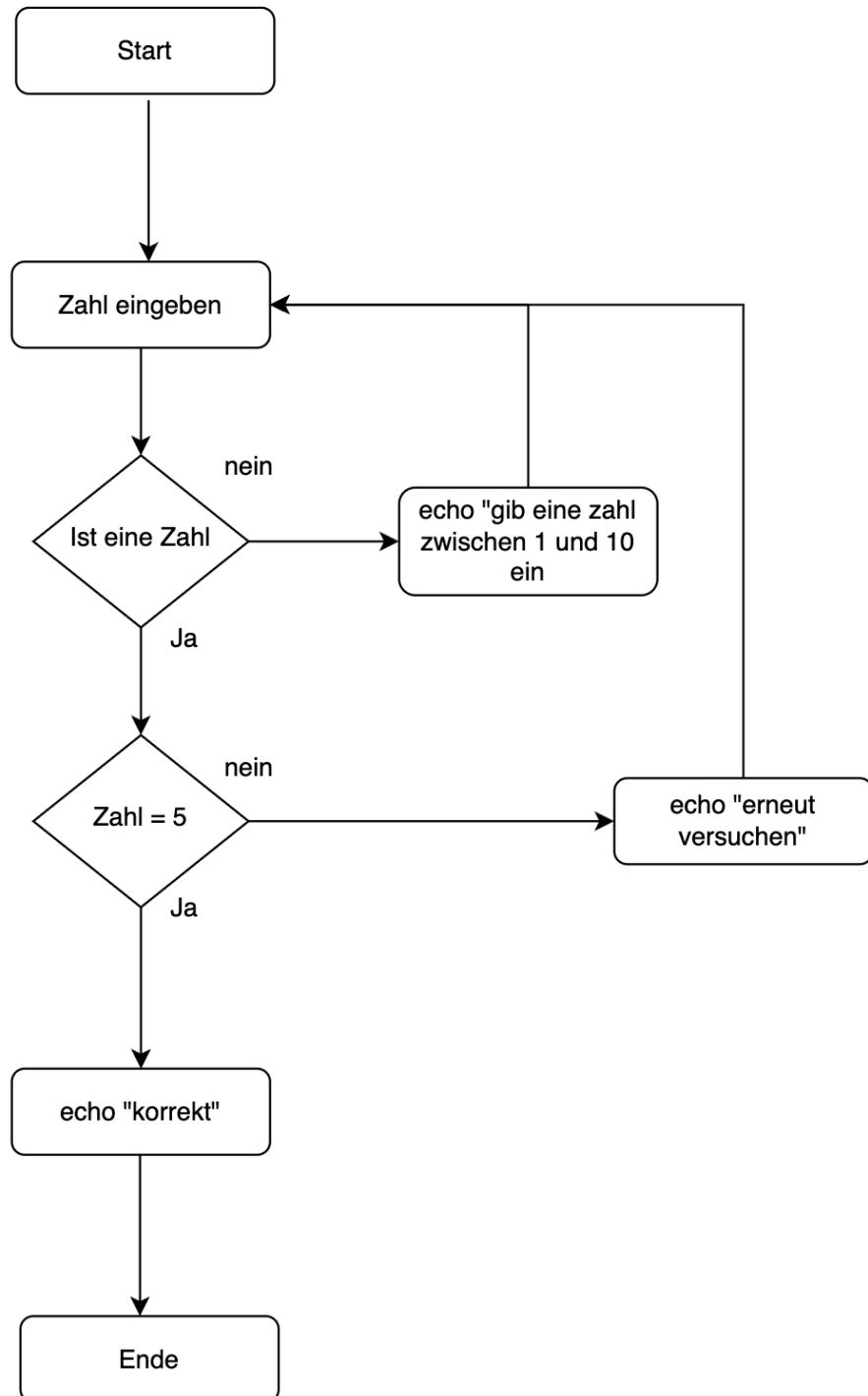
```
apple ➤ ~/Documents/schule/3Lehrjahr/M122/Zahlenraten ➤ ⚡ main ➤ sh zahlenraten.sh
Guess a number between 1 and 10:
100
Please enter a valid number between 1 and 10
```

Testfälle:

Tests werden mit der Zahl 5 durchgeführt, geraten kann nur bis zur Zahl 10

Testfall	Eingabe	Ausgabe
1	1	Too low. Guess again:
2	10	Too high. Guess again:
3	5	Congratulations! You guessed the number.
4	abc	Please enter a valid number between 1 and 10
5	11	Please enter a valid number between 1 and 10

Diagram:



Zahlenraten mit Rate Versuchen

Github:

- Script:

https://github.com/lcmoc/M122_bash_scripts/blob/0f5835daf046ae492496b146399fee02639e673e/Zahlenraten/zahlenraten.sh

- Diagram:

https://github.com/lcmoc/M122_bash_scripts/blob/main/Zahlenraten/bashScript.drawio

Script:

```
#!/bin/bash

number=$((RANDOM % 10 + 1))
attempts=3 # Anzahl der Versuche
guess_count=0

echo "Guess a number between 1 and 10:"

while [ $guess_count -lt $attempts ]; do
    echo "made guesses: $guess_count"
    echo "max guesses: $attempts"
    echo "-----"
    read guess

    if ! [[ "$guess" =~ ^[0-9]+\$ ]] || [ "$guess" -gt 10 ]; then
        echo "Please enter a valid number between 1 and 10"
        guess_count=$((guess_count+1))
        continue
    fi

    if [ "$guess" -lt "$number" ]; then
        echo "Too low. Guess again:"
        guess_count=$((guess_count+1))
        continue
    fi

    if [ "$guess" -gt "$number" ]; then
        echo "Too high. Guess again:"
        guess_count=$((guess_count+1))
        continue
    fi

    echo "Congratulations! You guessed the number."
    break
done

if [ $guess_count -eq $attempts ]; then
    echo "Sorry, you've run out of attempts. The number was $number."
fi
```

Ausgabe:

```
apple ~ /Doc/schule/3/M12/Zahlenraten ➜ ✎ main !1 ?6 ➜ sh zahlenratenRateversuchen.sh
Guess a number between 1 and 10:
made guesses: 0
max guesses: 3
-----
1
Too low. Guess again:
made guesses: 1
max guesses: 3
-----
2
Too low. Guess again:
made guesses: 2
max guesses: 3
-----
3
Too low. Guess again:
Sorry, you've run out of attempts. The number was 10.

apple ~ /Doc/schule/3/M12/Zahlenraten ➜ ✎ main !1 ?6 ➜
```

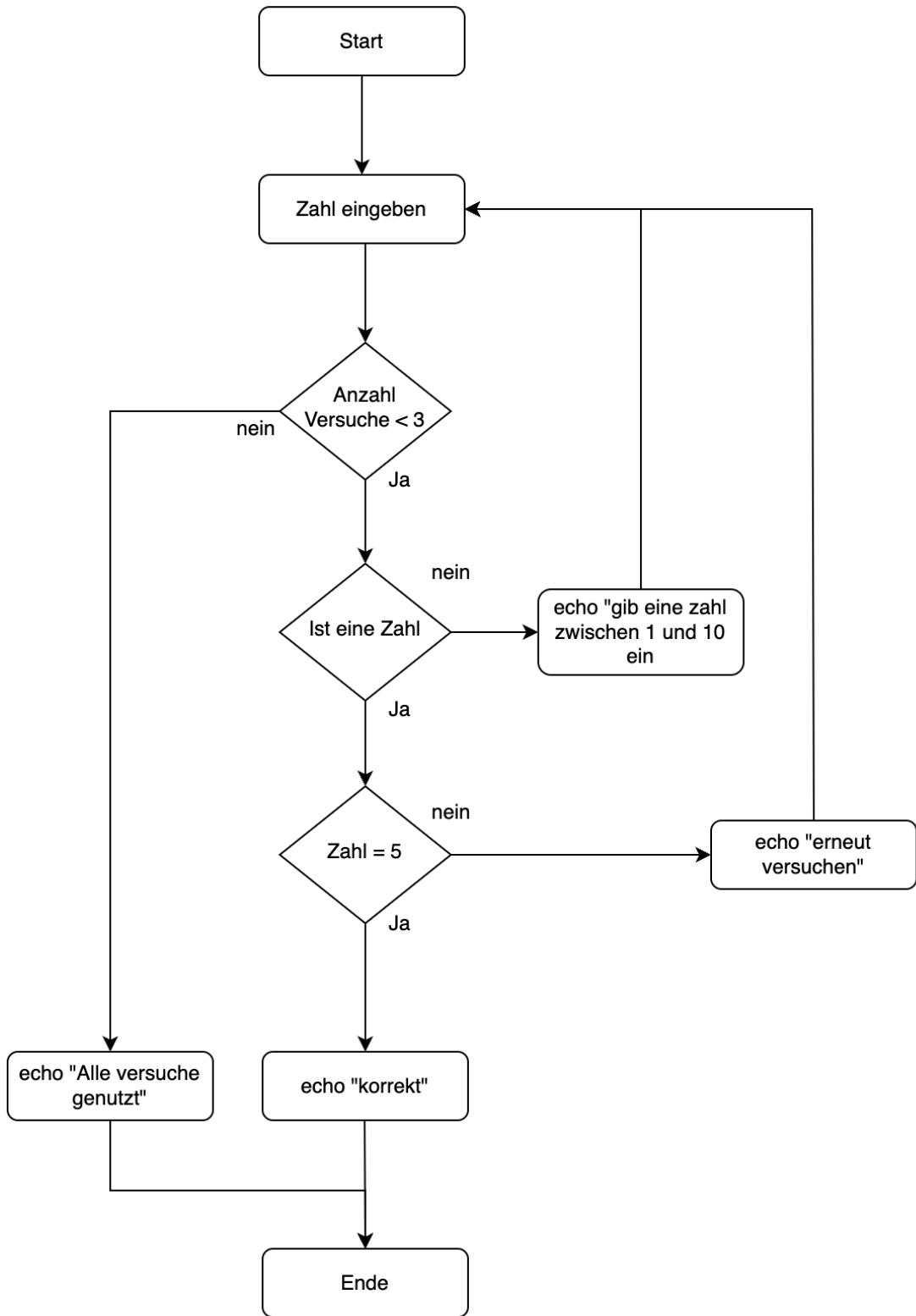
```
apple ~ /Documents/schule/3Lehrjahr/M122/Zahlenraten ➜ ✎ main ➜ sh zahlenraten.sh
Guess a number between 1 and 10:
100
Please enter a valid number between 1 and 10
```

Testfälle:

Tests werden mit der Zahl 5 durchgeführt, geraten kann nur bis zur Zahl 10, anzahl Versuche 3

Testfall	Eingabe	Ausgabe
1	1	Too low. Guess again:
2	10	Too high. Guess again:
3	5	Congratulations! You guessed the number.
4	abc	Please enter a valid number between 1 and 10
5	11	Please enter a valid number between 1 and 10
6	Zu viele eingaben	Sorry, you've run out of attempts. The number was 5

Diagram:



Zusatz

Github:

- Script:

https://github.com/lcmoc/M122_bash_scripts/blob/0f5835daf046ae492496b146399fee02639e673e/Zahlenraten/zahlenratenSchwierigkeit.sh

Bei diesem Code kann der Nutzer zusätzlich auch noch die Schwierigkeit anhand der Anzahl Versuche wählen.

Script:

```
#!/bin/bash

echo "Choose the difficulty level:"
echo "1. Easy (10 attempts)"
echo "2. Medium (7 attempts)"
echo "3. Hard (5 attempts)"
read difficulty

case $difficulty in
  1) attempts=10;;
  2) attempts=7;;
  3) attempts=5;;
 *) echo "Invalid option selected. Defaulting to easy difficulty (10 attempts)."
    attempts=10;;
esac

number=$((RANDOM % 10 + 1))
guess_count=0

echo "Guess a number between 1 and 10. You have $attempts attempts:"

while [ $guess_count -lt $attempts ]; do
  echo "made guesses: $guess_count"
  echo "max guesses: $attempts"
  echo "-----"
  read guess

  if ! [[ "$guess" =~ ^[0-9]*$ ]] || [ "$guess" -gt 10 ]; then
    echo "Please enter a valid number between 1 and 10"
    guess_count=$((guess_count+1))
    continue
  fi

  if [ "$guess" -lt "$number" ]; then
    echo "Too low. Guess again!"
    guess_count=$((guess_count+1))
    continue
  fi

  if [ "$guess" -gt "$number" ]; then
    echo "Too high. Guess again!"
    guess_count=$((guess_count+1))
    continue
  fi

  echo "Congratulations! You guessed the number."
  break
done

if [ $guess_count -eq $attempts ]; then
  echo "Sorry, you've run out of attempts. The number was $number."
fi
```

Ausgabe:

```
▶ ~/Doc/schule/3Lehrjahr/M122/Zahlenraten ▶ ↵ ⚡ main ?1 ▶ sh zahlenratenSchwierigkeit.sh
Choose the difficulty level:
1. Easy (10 attempts)
2. Medium (7 attempts)
3. Hard (5 attempts)
3
Guess a number between 1 and 10. You have 5 attempts:
made guesses: 0
max guesses: 5
-----
```

Aufgabe 2 Achtung Umleitung

ls -l

Dieser Befehl listet alle Files in dem Repository, in welchem man sich befindet, auf und zeigt sie als Liste an.

Ausgabe:

```
 MacBook-Pro ~ % cd ~/Doc/schule/3Lehrjahr/M122/AchtungUmleitung % ls -l
total 0
-rw-r--r-- 1 mcl  staff  0 16 Mär 16:04 test1.txt
-rw-r--r-- 1 mcl  staff  0 16 Mär 16:04 test2.txt
-rw-r--r--@ 1 mcl  staff  0 16 Mär 16:04 test3.txt
-rw-r--r-- 1 mcl  staff  0 16 Mär 16:05 test4.txt
```

ls -l > ls-l.txt

Dieser Befehl listet alle Files in dem Repository, in welchem man sich befindet, auf und zeigt sie als Liste an.

Ausgabe:

```
 1 total 0
 2 -rw-r--r-- 1 mcl  staff  0 16 Mär 16:12 ls-l.txt
 3 -rw-r--r-- 1 mcl  staff  0 16 Mär 16:04 test1.txt
 4 -rw-r--r-- 1 mcl  staff  0 16 Mär 16:04 test2.txt
 5 -rw-r--r--@ 1 mcl  staff  0 16 Mär 16:04 test3.txt
 6 -rw-r--r-- 1 mcl  staff  0 16 Mär 16:05 test4.txt
 7
```

rmdir

Entfernt ein vorhandenes Verzeichnis und gibt eine Fehlermeldung aus, falls das angegebene Verzeichnis nicht existiert.

1 Fehlermeldung

Unset

```
rmdir VerzeichnisExistiertNicht
```

Da das Verzeichnis welches nicht existiert gelöscht werden soll, wird eine Fehlermeldung ausgegeben, welche beschreibt, dass weder ein File noch ein Ordner mit diesem Namen existiert

Ausgabe:

```
apple ~ /Doc/schule/3Lehrjahr/M122/AchtungUmleitung ➜ ↵ p main ?2 ➜ ls -l
total 8
-rw-r--r-- 1 mcl staff 267 16 Mär 16:12 ls-l.txt
-rw-r--r-- 1 mcl staff 0 16 Mär 16:04 test1.txt
-rw-r--r-- 1 mcl staff 0 16 Mär 16:04 test2.txt
-rw-r--r--@ 1 mcl staff 0 16 Mär 16:04 test3.txt
-rw-r--r-- 1 mcl staff 0 16 Mär 16:05 test4.txt

apple ~ /Doc/schule/3Lehrjahr/M122/AchtungUmleitung ➜ ↵ p main ?2 ➜ rmkdir VerzeichnisExistiertNicht
rmkdir: VerzeichnisExistiertNicht: No such file or directory

apple ~ /Doc/schule/3/M122/AchtungUmleitung ➜ ↵ p main ?2 ➜
```

2 Fehlermeldung, log file wird erstellt

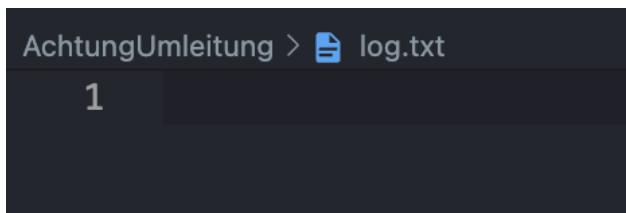
Da der Befehl, die Fehlermeldung auf die Konsole printed, wird diese nicht in dem, durch den Befehl erstellen Datei, angezeigt

Unset

```
rmdir VerzeichnisExistiertNicht > log.txt
```

Ausgabe:

```
apple ~ /Doc/schule/3/M122/AchtungUmleitung ➜ main ?2 ➜ rmdir VerzeichnisExistiertNicht > log.txt  
rmdir: VerzeichnisExistiertNicht: No such file or directory
```



3 Fehlermeldung mit Logfile Erstellung

Mit diesem Befehl werden die Fehlermeldungen, die durch den ausgeführten Befehl entstehen, in das Logfile hineingeschrieben.

Durch die 2, wird die Fehlermeldung nicht mehr im Terminal angezeigt, sondern in das Logfile reingeschrieben

Unset

```
rmdir VerzeichnisExistiertNicht 2> errorlog.txt
```

Ausgabe:

```
apple ➤ ~/Doc/schule/3/M122/AchtungUmleitung ➤ ⚡ main ?2 ➤ rmdir VerzeichnisExistiertNicht 2> log2.txt
apple ➤ ~/Doc/schule/3/M122/AchtungUmleitung ➤ ⚡ main ?2 ➤
```

```
log2.txt
1  rmdir: VerzeichnisExistiertNicht: No such file or directory
2  |
```

ls -l | grep ".\txt"

Beschreibung von der von der Vorherigen Aufgabe:

Dieser Befehl listet alle Files in dem Repository, in welchem man sich befindet, auf und zeigt sie als Liste an.

Unset

```
ls -l
```

Ausgabe:

```
total 16
-rw-r--r-- 1 mcl staff 0 16 Mär 16:35 log.txt
-rw-r--r-- 1 mcl staff 60 16 Mär 16:35 log2.txt
-rw-r--r-- 1 mcl staff 267 16 Mär 16:12 ls-l.txt
-rw-r--r-- 1 mcl staff 0 21 Mär 09:07 test.bat
-rw-r--r-- 1 mcl staff 0 20 Mär 14:40 test.doc
-rw-r--r-- 1 mcl staff 0 20 Mär 14:39 test.html
-rw-r--r-- 1 mcl staff 0 20 Mär 14:39 test.java
-rw-r--r-- 1 mcl staff 0 20 Mär 14:39 test.js
-rw-r--r-- 1 mcl staff 0 20 Mär 14:39 test.json
-rw-r--r-- 1 mcl staff 0 20 Mär 14:39 test.jsx
-rw-r--r-- 1 mcl staff 0 20 Mär 14:39 test.php
-rw-r--r-- 1 mcl staff 0 20 Mär 14:39 test.py
-rw-r--r--@ 1 mcl staff 0 20 Mär 14:40 test.sql
-rw-r--r-- 1 mcl staff 0 16 Mär 16:04 test1.txt
-rw-r--r-- 1 mcl staff 0 16 Mär 16:04 test2.txt
-rw-r--r--@ 1 mcl staff 0 16 Mär 16:04 test3.txt
-rw-r--r-- 1 mcl staff 0 16 Mär 16:05 test4.txt
```

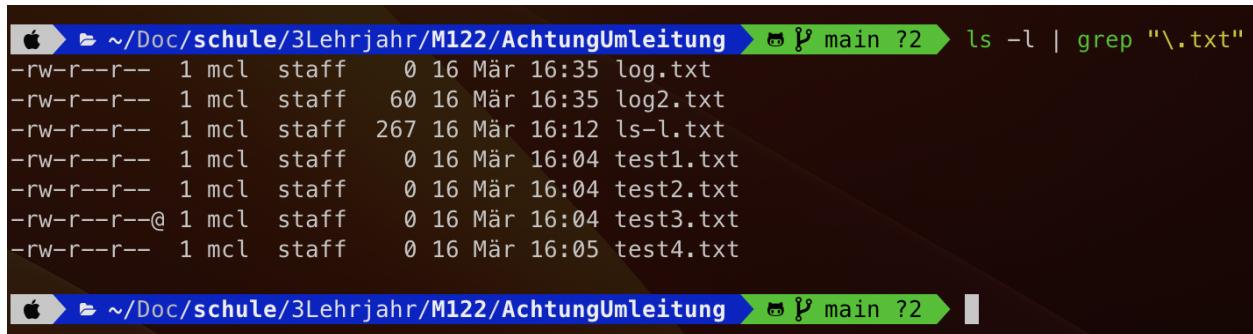
Das "|" Symbol wird dafür verwendet, den Output eines Kommandos direkt als Input für das nächste Kommando in einer Befehlskette zu übergeben. Damit lässt sich also direkt eine Bedingung für den vorherigen Befehl hinzufügen.

Der Befehl wird nur Text Dateien ausführen, da man durch den Befehl "grep" nach bestimmten Formationen oder Inhalten filtern kann.

Unset

```
ls -l | grep "\.txt"
```

Ausgabe:



```
apple ➤ ~/Doc/schule/3Lehrjahr/M122/AchtungUmleitung ➤ ↵ P main ?2 ➤ ls -l | grep "\.txt"
-rw-r--r-- 1 mcl staff 0 16 Mär 16:35 log.txt
-rw-r--r-- 1 mcl staff 60 16 Mär 16:35 log2.txt
-rw-r--r-- 1 mcl staff 267 16 Mär 16:12 ls-l.txt
-rw-r--r-- 1 mcl staff 0 16 Mär 16:04 test1.txt
-rw-r--r-- 1 mcl staff 0 16 Mär 16:04 test2.txt
-rw-r--r--@ 1 mcl staff 0 16 Mär 16:04 test3.txt
-rw-r--r-- 1 mcl staff 0 16 Mär 16:05 test4.txt
```

Um jetzt z.B nur die Bat Dateien zu erhalten müsste man den folgenden Befehl ausführen.

Unset

```
ls -l | grep "\.bat"
```

Ausgabe:

```
apple ➔ ~/Doc/schule/3Lehrjahr/M122/AchtungUmleitung ➔ cat main ?2 ➔ ls -l | grep "\.bat"
-rw-r--r-- 1 mcl staff 0 21 Mär 09:07 test.bat
```

Andere Beispiele:

```
apple ➔ ~/Doc/schule/3Lehrjahr/M122/AchtungUmleitung ➔ cat main ?2 ➔ ls -l | grep "\.json"
-rw-r--r-- 1 mcl staff 0 20 Mär 14:39 test.json
```

```
apple ➔ ~/Doc/schule/3Lehrjahr/M122/AchtungUmleitung ➔ cat main ?2 ➔ ls -l | grep "\.java"
-rw-r--r-- 1 mcl staff 0 20 Mär 14:39 test.java
```

Wichtig zu beachten ist auch, dass "grep" ein Pattern als Parameter nimmt, was heißt, dass wenn man das gewünschte Format nicht genauer ausgrenzt beziehungsweise definiert, könnte man, wie bei diesem Beispiel, nicht gewünschte Files erhalten.

```
apple ➔ ~/Doc/schule/3Lehrjahr/M122/AchtungUmleitung ➔ cat main ?2 ➔ ls -l | grep "\.js"
-rw-r--r-- 1 mcl staff 0 20 Mär 14:39 test.js
-rw-r--r-- 1 mcl staff 0 20 Mär 14:39 test.json
-rw-r--r-- 1 mcl staff 0 20 Mär 14:39 test.jsx
```

Ich wollte in diesem Fall nur js files erhalten, da aber "json" Files und "jsx" Files, ebenfalls mit "js" beginnen, wurden auch diese aufgelistet. Um das zu beheben kann der folgende Befehl ausgeführt werden:

Unset

```
ls -l | grep "\.js$"
```

Ausgabe:

```
apple ➔ ~/Doc/schule/3Lehrjahr/M122/AchtungUmleitung ➔ cat main ?2 ➔ ls -l | grep "\.js$"
-rw-r--r-- 1 mcl staff 0 20 Mär 14:39 test.js
```

Das \$ Zeichen gibt in dem Fall an, dass die Suche am Ende der Zeile erfolgen soll. Dadurch wird sichergestellt, dass nur die Zeilen mit dem exakten Muster ".js" angezeigt werden.

ls -l | sed -e "s/[aeio]/X/g"

Das Kommando "[aeio]" ersetzt die Buchstaben a, e, i und o - durch das Zeichen "X".

Das "g" steht für global und bewirkt, dass die Ersetzung auf alle Vorkommnisse in jeder Zeile angewendet wird.

Das heißt, in der Auflistung mit ls sind die Umlaute a, e, i und o nicht vorhanden, beziehungsweise wurden durch ein X ersetzt.

Unset

ls -l | sed -e "s/[aeio]/X/g"

Ausgabe:

```
apple ➤ ~/Doc/schule/3/M122/AchtungUmleitung ➤ ↵ ⌘ P main ?2 ➤ ls -l | sed -e "s/[aeio]/X/g"

tXtXl 16
-rw-r--r-- 1 mcl stXff 0 16 Mär 16:35 lXg.txt
-rw-r--r-- 1 mcl stXff 60 16 Mär 16:35 lXg2.txt
-rw-r--r--@ 1 mcl stXff 267 16 Mär 16:12 ls-l.txt
-rw-r--r-- 1 mcl stXff 0 21 Mär 09:07 tXst.bXt
-rw-r--r-- 1 mcl stXff 0 20 Mär 14:40 tXst.dXc
-rw-r--r-- 1 mcl stXff 0 20 Mär 14:39 tXst.html
-rw-r--r-- 1 mcl stXff 0 20 Mär 14:39 tXst.jXvX
-rw-r--r-- 1 mcl stXff 0 20 Mär 14:39 tXst.js
-rw-r--r-- 1 mcl stXff 0 20 Mär 14:39 tXst.jsXn
-rw-r--r-- 1 mcl stXff 0 20 Mär 14:39 tXst.jsx
-rw-r--r-- 1 mcl stXff 0 20 Mär 14:39 tXst.php
-rw-r--r-- 1 mcl stXff 0 20 Mär 14:39 tXst.py
-rw-r--r--@ 1 mcl stXff 0 20 Mär 14:40 tXst.sql
-rw-r--r-- 1 mcl stXff 0 16 Mär 16:04 tXst1.txt
-rw-r--r-- 1 mcl stXff 0 16 Mär 16:04 tXst2.txt
-rw-r--r--@ 1 mcl stXff 0 16 Mär 16:04 tXst3.txt
-rw-r--r-- 1 mcl stXff 0 16 Mär 16:05 tXst4.txt
```

Cat

Der Befehl "cat" wird dafür verwendet, um Inhalte einer Datei anzuzeigen.

Unset

```
cat test1.txt
```

Ausgabe:

```
test1.txt
1 This is the test1 text file
2 |
```

```
~/.Doc/schule/3Lehrjahr/M122/AchtungUmleitung ➔ main ?2 ➔ cat test1.txt
This is the test1 text file

~/.Doc/schule/3Lehrjahr/M122/AchtungUmleitung ➔ main ?2 ➔
```

1 Fehlermeldung

Beschreibung:

Wie bei "rmdir", wird bei dem Befehl "cat" ebenfalls eine Fehlermeldung ausgegeben, falls die Datei, welche man anzeigen will, nicht vorhanden ist.

Unset

```
cat dateiDieEsNichtGibt.txt
```

Ausgabe:

```
apple ~ /Doc/schule/3Lehrjahr/M122/AchtungUmleitung ➜ cat dateiDieEsNichtGibt.txt  
cat: dateiDieEsNichtGibt.txt: No such file or directory  
apple ~ /Doc/schule/3/M122/AchtungUmleitung ➜
```

2 Fehlermeldung mit Logfile Erstellung

Wie beim “rmdir” beispiel wird hier ebenfalls eine Datei erstellt, welche die Fehlermeldung enthält

Unset

```
cat dateiDieEsNichtGibt.txt 2> error3.txt
```

Ausgabe:

```
apple ~ /Doc/schule/3Lehrjahr/M122/AchtungUmleitung ➜ cat dateiDieEsNichtGibt.txt 2> error3.txt  
apple ~ /Doc/schule/3/M122/AchtungUmleitung ➜
```

```
AchtungUmleitung > error3.txt  
1  cat: dateiDieEsNichtGibt.txt: No such file or directory  
2  |
```

3 Fehlermeldung im Blackhole

Bei dem folgenden Befehl wird keine Fehlermeldung auf die Konsole geschrieben. Die entsprechende Meldung landet in einem sogenannten Black Hole, in welchem Daten direkt gelöscht werden. Der Pfad dafür lauten /dev/null

Unset

```
cat dateiDieEsNichtGibt.txt 2> /dev/null
```

Ausgabe:

```
apple ➔ ~/Doc/schule/3/M122/AchtungUmleitung ➔ ⌘ P main ?1 ➔ cat dateiDieEsNichtGibt.txt  
cat: dateiDieEsNichtGibt.txt: No such file or directory  
  
apple ➔ ~/Doc/schule/3/M122/AchtungUmleitung ➔ ⌘ P main ?1 ➔
```

Aufgabe 3 Du Bytes

Der Befehl "du" ist ein Kürzel für "Disk Usage" und wird in Unix-basierten Betriebssystemen wie Mac OS verwendet, um Informationen über die Speichernutzung von Dateien und Verzeichnissen anzuzeigen.

Wenn der Befehl "du" alleine verwendet wird, wird er den Speicherplatzbedarf aller Dateien und Verzeichnisse im aktuellen Verzeichnis einschließlich aller Unterverzeichnisse anzeigen.

Die erste Zahl "8" gibt die Größe des aktuellen Verzeichnisses in Kilobyte (KB) an, während das "." das aktuelle Verzeichnis selbst darstellt.

```
Unset
```

```
du
```

Ausgabe:

```
8
```

```
8
```

```
test.txt U X
```

```
DuBytes > test.txt
```

```
1 das ist ein text|
```

--apparent-size:

(Im mac terminal nicht verfügbar)

Mit dieser Option zeigt "du" die "scheinbare Größe" der Dateien an, d.h. die tatsächliche Größe jeder Datei auf der Festplatte, ohne Berücksichtigung der Komprimierung oder Deduplizierung von Daten. In einigen Fällen kann die scheinbare Größe von der tatsächlichen Größe der Datei abweichen, wenn die Datei komprimiert ist oder Sparse-Dateien verwendet.

--block-size=1:

(Im mac terminal nicht verfügbar)

Mit dieser Option zeigt du die Größe jedes Datei-Blocks in Byte an. Normalerweise verwendet "du" eine Blockgröße, die vom Betriebssystem festgelegt wird und größer als 1 Byte ist, was dazu führen kann, dass die gemeldete Größe einer Datei größer ist als die tatsächliche Größe der Datei. Wenn Sie jedoch --block-size=1 verwenden, zeigt "du" die genaue Größe jeder Datei in Byte an. Diese Option kann nützlich sein, um die Größe von sehr kleinen Dateien genau zu messen oder um genauere Berechnungen durchzuführen, wenn Sie "du" mit anderen Befehlen verknüpfen möchten.

Datei auflistung

Um alle Dateien und Verzeichnisse in Bytes aufzulisten, kann der Folgende Befehl verwendet werden:

Unset

```
du -s -k * | awk '{print $1*1024, "\t", $2}'
```

Ausgabe:



```
4096    gugus.txt
4096    test.txt
4096    testfolder
```

Dieser Befehl gibt eine Liste aller Dateien und Verzeichnisse im aktuellen Verzeichnis aus, zusammen mit der Größe jedes Elementes in Bytes. Die Option "-s" gibt an, dass nur eine Zusammenfassung der Größe für jedes Element ausgegeben werden soll, während "-k" dafür sorgt, dass die Größen in Kilobytes angezeigt werden. Der Befehl wird anschließend mithilfe des "awk" Befehls verarbeitet, der die Größe jedes Elementes in Kilobytes in Bytes umrechnet. Die Ausgabe enthält schließlich die Größe jedes Elements in Bytes sowie den Namen des Elements.

Summe ausgeben

Um die Summe aller Dateien und Verzeichnisse in einem Verzeichnis mit dem "du" Befehl auszugeben, können Sie die Option "-s" oder "--summarize" verwenden.

```
Unset
```

```
du -s
```

Ausgabe:

```
➔ ~/Documents/schule/3Lehrjahr/M122/DuBytes ➔ ⚡ p main ?3 ➔ du -s  
24 .
```

Wenn man die Größe der Dateien auch noch in "menschen lesbarer" Version haben will, kann man die Option "-h" oder "--human-readable" hinzufügen.

```
Unset
```

```
du -sh
```

Ausgabe:

```
➔ ~/Documents/schule/3Lehrjahr/M122/DuBytes ➔ ⚡ p main ?3 ➔ du -sh  
12K .
```

Aufgabe mit exec

Um nur die Größen der Files mit “du” zu erhalten, kann Folgender Code ausgeführt werden.

```
Unset
```

```
find . -type f -exec du -a {} +
```

Ausgabe:



```
MacBook-Pro: ~/Doc/schule/3/M122/DuBytes ➜ main ?3 ➔ find . -type f -exec du -a {} +  
8 ./testfolder/blabla.txt  
8 ./gugus.txt  
8 ./test.txt
```

Für jede Datei, die durch den Befehl “find” gefunden wird, führt der Befehl “du -a {}” aus, um die Größe der Datei auszugeben.

Die Option -a gibt dabei die Größe jeder einzelnen Datei aus, inklusive der Größe von Unterverzeichnissen, die sich innerhalb der Datei befinden können.

Das + am Ende der “exec-Option” ist eine Alternative zum Semikolon “;”. Es signalisiert, dass die du-Befehle in größeren Gruppen (statt einzeln) ausgeführt werden sollen, um die Effizienz zu verbessern. Mit anderen Worten, anstatt für jede einzelne Datei den du-Befehl aufzurufen, werden mehrere Dateien zu einer Gruppe zusammengefasst und der du-Befehl wird einmal für die gesamte Gruppe ausgeführt.

Um das Ergebnis auch noch der Grösse nach Sortiert zu erhalten, kann man zusätzlich noch “| sort -n -r” hinzufügen

```
Unset
```

```
find . -type f -exec du -a {} + | sort -n -r
```

Ausgabe:

```
8     ./testfolder/blabla.txt
8     ./test.txt
8     ./gugus.txt
```

“sort” ist ein Befehl, der die Eingabe sortiert und auf dem Bildschirm ausgibt. Die Option -n gibt an, dass die Sortierung numerisch erfolgen soll, d.h. nach Größe der Dateien. Die Option -r gibt an, dass die Sortierung umgekehrt erfolgen soll, d.h. von der größten zur kleinsten Größe.

Script duBytesSuender.sh

Github:

- Script

https://github.com/lcmoc/M122_bash_scripts/blob/0f5835daf046ae492496b146399fee02639e673e/DuBytes/duBytesSuender.sh

Script:

```
DuBytes > duBytesSuender.sh
1 #!/bin/bash
2
3 # calculate total size of all files in bytes
4 total_size=$(find . -type f -print0 | xargs -0 stat -f "%z" | awk '{sum+=$1} END {print sum}')
5
6 # print total size in human-readable format
7 echo "Total size: $(echo $total_size | awk '{ split( \"B KB MB GB TB\" , v ); s=1; while( $1>=1024 ) { $1/=1024; s++ } printf \"%.\$2f %s\\n\" , \$1, v[s] }')"
8
9 # list top 3 files by size
10 find . -type f -exec ls -lh {} + | awk '{print $5, $9}' | sort -hr | head -n 3 | awk '{print NR ". " $2 " (" $1 ")"}'
11 |
```

Ausgabe:

```
Total size: 1,79 KB
1. ./testfolder/blabla.txt (1,3K)
2. ./duBytesSuender.sh (494B)
3. ./test.txt (31B)
```

find . -type f -print0

Dieser Befehl sucht nach allen Dateien im aktuellen Verzeichnis (".") und gibt sie als Null-terminierte Zeichenfolge aus. Das Nullzeichen wird verwendet, um sicherzustellen, dass der Befehl mit Dateinamen umgehen kann, die Leerzeichen oder Sonderzeichen enthalten.

xargs -0 stat -f "%z"

Dieser Befehl nimmt die Ausgabe des find-Befehls (alle Dateien im aktuellen Verzeichnis und den darin enthaltenen Unterverzeichnissen) als Eingabe und führt den Befehl "stat" für jede Datei aus, um ihre Größe in Bytes abzurufen. Der Parameter "-f" wird verwendet, um das Format der Ausgabe von "stat" anzugeben, und "%z" gibt die Größe der Datei in Bytes zurück.

```
awk '{sum+=$1} END {print sum}'
```

Dieser Befehl wird auf die Ausgabe von xargs angewendet, um die Größen aller Dateien zu summieren. Der Befehl "awk" durchläuft jede Zeile der Eingabe und addiert die Größe der Datei (die erste Spalte in der Ausgabe von stat) zur Variable "sum". Am Ende wird die Gesamtgröße der Dateien in Byte ausgegeben.

```
echo "Total size: $(echo $total_size | awk '{ split( "B KB MB GB TB" , v ); s=1; while( $1>=1024 ){ $1/=1024; s++ } printf "%..2f %s\n", $1, v[s] }")"
```

Dieser Befehl gibt die Gesamtgröße der Dateien in einer für Menschen lesbaren Form aus. Die Größe wird in KB, MB, GB oder TB angegeben, je nachdem, wie groß die Größe ist. Dabei wird die Variable "total_size" (die die Gesamtgröße der Dateien in Byte enthält) verwendet. Der Befehl "awk" teilt die Größe durch 1024, bis sie kleiner als 1024 ist, und wählt dann die entsprechende Einheit aus ("B", "KB", "MB", "GB" oder "TB").

```
find . -type f -exec ls -lh {} + | awk '{print $5, $9}' | sort -hr | head -n 3 | awk '{print NR ". " $2 " (" $1 ")"}'
```

Dieser Befehl listet die drei größten Dateien im aktuellen Verzeichnis und den darin enthaltenen Unterverzeichnissen auf, sortiert nach Größe und formatiert als Rangliste mit Dateiname und Größe. Der Befehl "find" sucht nach allen Dateien im aktuellen Verzeichnis und den darin enthaltenen Unterverzeichnissen. Der Befehl "ls" wird für jede Datei ausgeführt, um ihre Größe und den Dateinamen anzuzeigen, und die Ausgabe wird an "awk" weitergegeben, um nur die Größe und den Dateinamen auszuwählen. Der Befehl "sort" sortiert die Dateien nach Größe, und "head" wählt die drei größten aus. Schließlich wird "awk" verwendet, um die Dateien als Rangliste mit Dateiname und Größe zu formatieren.

Script duBytesSuender.sh Erweiterung

Github:

- Script:

https://github.com/lcmoc/M122_bash_scripts/blob/0f5835daf046ae492496b146399fee02639e673e/duBytesSuenderErweiterung.sh

Script:

```
duBytesSuenderErweiterung.sh
1 #!/bin/bash
2
3 if [ $# -eq 0 ]; then
4   echo "Usage: $0 <directory>"
5   echo "Dieses Script listet ab einem Verzeichnis rekursiv die summe des Speicherbedarfs auf."
6   echo "Sodann noch die drei groesten Dateien."
7   echo ""
8   echo "Sie muessen beim Aufruf des Scripts $0 den Namen des geünschten Verzeichnis mitgeben."
9   exit 1
10 fi
11
12 if [ ! -d "$1" ]; then
13   echo "Error: Directory '$1' does not exist"
14   exit 1
15 fi
16
17 total_size=$(find "$1" -type f -print0 | xargs -0 stat -f "%z" | awk '{sum+=$1} END {print sum}')
18
19 echo "Total size: $(echo $total_size | awk '{ split( \"B KB MB GB TB\" , v ); s=1; while( $1>=1024 ){ $1/=1024; s++ } printf \"%,.2f %s\n\", $1, v[s] }')"
20
21 find "$1" -type f -exec ls -lh {} + | awk '{print $5, $9}' | sort -hr | head -n 3 | awk '{print NR ". " $2 " (" $1 ")"}'
22
```

Ausgabe:

```
~/Documents/schule/3Lehrjahr/M122 ➤ main ➤ sh duBytesSuenderErweiterung.sh DuBytes
Total size: 1,68 KB
1. DuBytes/testfolder/blabla.txt (1,3K)
2. DuBytes/duBytesSuender.sh (378B)
3. DuBytes/test.txt (31B)
```

Dieses Skript nimmt ein Verzeichnis als Input und listet dabei die größten 3 Dateien im angegebenen Verzeichnis mit dessen Unterverzeichnissen auf. Ebenfalls wird, falls kein Verzeichnis angegeben wird, eine Beschreibung wie man den Befehl anwendet ausgegeben. Zudem erhält man auch noch eine Fehlermeldung, falls das angegebene Verzeichnis nicht existiert.

```
~/Documents/schule/3Lehrjahr/M122 ➤ main +1 !1 ?2 ➤ sh duBytesSuenderErweiterung.sh
Usage: duBytesSuenderErweiterung.sh <directory>
Dieses Script listet ab einem Verzeichnis rekursiv die summe des Speicherbedarfs auf.
Sodann noch die drei groesten Dateien.

Sie muessen beim Aufruf des Scripts duBytesSuenderErweiterung.sh den Namen des geünschten Verzeichnis mitgeben.

~/Documents/schule/3Lehrjahr/M122 ➤ main +1 !1 ?2 ➤
```

```
 MacBook-Pro:~ LucaMock$ cd ~/Documents/schule/3Lehrjahr/M122
MacBook-Pro:~/Documents/schule/3Lehrjahr/M122 LucaMock$ ./nichtVorhandenesVerzeichnis.sh
Error: Directory 'nichtVorhandenesVerzeichnis' does not exist
MacBook-Pro:~/Documents/schule/3Lehrjahr/M122 LucaMock$
```

Script “duBytesFileSizeDecreasing” Zusatz

Github:

- duBytesFileSizeDecreasing:

https://github.com/lcmoc/M122_bash_scripts/blob/0f5835daf046ae492496b146399fee02639e673e/duBytesFileSizeDecreasing.sh

- RemoveComments:

https://github.com/lcmoc/M122_bash_scripts/blob/0f5835daf046ae492496b146399fee02639e673e/RemoveComments/removeComments.sh

Einleitung

Bei diesem Bash-Script geht es darum, dass die Größe aller Dateien in einem bestimmten Verzeichnis berechnet und die drei größten Dateien aufgelistet werden. Es fragt dann den Nutzer, ob er die Größe der größten Dateien reduzieren möchte, indem alle Kommentare und Leerzeichen entfernt werden. Wenn der Nutzer dies bestätigt, entfernt das Script die Kommentare und Leerzeichen aus den drei größten Dateien und zeigt die neuen Größen an. Das Ziel des Projekts ist es, die Verwaltung von großen Dateien zu erleichtern, indem unnötiger Code entfernt wird, um Speicherplatz zu sparen.

Bisher sind nur die Dateiformate js und sh implementiert, dies kann aber beliebig ausgebaut werden.

Um das ganze umzusetzen, habe ich erstmal ein Script geschrieben, welches Kommentare und Leerzeichen von einem angegebenen File entfernt. Siehe Github RemoveComments

Script:

```
You, 2 minutes ago | 1 author (You)
1 #!/bin/bash
2
3 # Prüfe, ob eine Datei angegeben wurde
4 if [ -z "$1" ]; then
5   echo "Bitte geben Sie eine Datei an."
6   exit 1
7 fi
8
9 # Prüfe, ob die Datei existiert
10 if [ ! -f "$1" ]; then
11   echo "Die Datei $1 existiert nicht."
12   exit 1
13 fi
14
15 # Entferne Kommentare und leere Zeilen aus der Datei
16 if [[ "$1" == *.sh ]]; then
17   sed -i '' -e '/^#*/d' -e '/^[:space:]*$/d' "$1"
18 elif [[ "$1" == *.js ]]; then
19   sed -i '' -e '/^#*/d' -e '/^[:space:]*$/d' "$1"
20 else
21   echo "Kommentare können nur aus .sh- oder .js-Dateien entfernt werden."
22   exit 1
23 fi
24
25 echo "Kommentare und leere Zeilen wurden aus der Datei $1 entfernt."
```

```
 MacBook-Pro ~ /Doc/schule/3Lehrjahr/M122/RemoveComments ➜ ⌘ P main ➜ sh removeComments.sh removeCommentsTest.js
Kommentare und leere Zeilen wurden aus der Datei removeCommentsTest.js entfernt.

MacBook-Pro ~ /Doc/schule/3Lehrjahr/M122/RemoveComments ➜ ⌘ P main ➜
```

“RemoveCommentsTest.js” File vor dem Ausführen des Skriptes:

```
1 // test
2
3 const a = 1;
4 const b = 2;
5
6 // test 2
7
8 const c = a + b;
9
10 // test 3
11
12 console.log("xxx", c);
13
14 // test 4
15 a;
16 |
```

“RemoveComments.js” Nach dem ausführen des removeComments.sh Skriptes:

```
You, 21 hours ago | 1 author (You)
1 const a = 1;
2 const b = 2;
3 const c = a + b;
4 console.log("xxx", c);
5 |
```

Da dieses Script / Funktion soweit funktionierte, musste ich das vorherige Script mit welchem ich die größten 3 Sünder Files ausfindig machen kann, nur noch mit diesem Script verbinden und eine User Interaktion mit einbauen. Das Ergebnis sehen Sie in den Bildern darunter, oder auf Github, den Link dazu habe ich bereits am Anfang dieser Zusatzaufgabe geteilt (Github “duBytesFileSizeDecreasing”).

Script:

```
duBytesFileSizeDecreasing.sh
1 #!/bin/bash
2
3 if [ $# -eq 0 ]; then
4   echo "Usage: $0 <directory>"
5   echo "Dieses Script listet ab einem Verzeichnis rekursiv die summe des Speicherbedarfs auf."
6   echo "Sodann noch die drei grössten Dateien."
7   echo ""
8   echo "Sie müssen beim Aufruf des Scripts $0 den Namen des gewünschten Verzeichnisses mitgeben."
9   exit 1
10 fi
11
12 if [ ! -d "$1" ]; then
13   echo "Fehler: Das Verzeichnis '$1' existiert nicht"
14   exit 1
15 fi
16
17 total_size=$(find "$1" -type f -print0 | xargs -0 stat -f "%z" | awk '{sum+=$1} END {print sum}')
18
19 echo "Gesamtgröße: $(echo $total_size | awk '{ split( \"B KB MB GB TB\" , v ); s=1; while( $1>1024 ){ $1/=1024; s++ } printf \"%.\", s, v[s] }')"
20
21 echo ""
22
23 largest_files=$(find "$1" -type f -exec ls -lh {} + | awk '{print $5, $9}' | sort -hr | head -n 3 | awk '{print NR ". " $2 " (" $1 ")"}')
24 echo "Die drei größten Dateien sind:"
25
26 echo ""
27
28 echo "$largest_files"
29
30 echo ""
31
32 read -p "Möchten Sie die Größe der Dateien verkleinern? (Ja/Nein) " choice
33 echo ""
34 case "$choice" in
35   j|J|ja|Ja|JA )
36     echo "Entferne Kommentare und Leerzeichen aus den Dateien.."
37     echo ""
38     for file in $(echo "$largest_files" | awk '{print $2}'); do
39       if [ -f "$file" ]; then
40         if [[ "$file" == *.sh ]]; then
41           sed -i '' -e '/^s*/d' -e '/^[:space:]]*$/d' "$file"
42         elif [[ "$file" == *.js ]]; then
43           sed -i '' -e '/^s*\/\//.*$/d' -e '/^[:space:]]*$/d' "$file"
44         else
45           echo "Kommentare können nur aus .sh- oder .js-Dateien entfernt werden."
46           exit 1
47         fi
48         new_size=$((stat -f "%z" "$file"))
49         echo "Neue Größe von '$file': $(numfmt --to=iec-i --suffix=B $new_size)"
50       else
51         echo "Fehler: Die Datei '$file' existiert nicht"
52       fi
53     done
54   ;;
55   n|N|nein|Nein|NEIN )
56     echo "Die Größe der Dateien wird nicht verkleinert."
57   ;;
58   * )
59     echo "Ungültige Antwort. Die Größe der Dateien wird nicht verkleinert."
60   ;;
61 esac
```

Sobald man das Script wie folgt ausführt (um es selbst zu testen können Sie meine Vorbereiteten Files im “RemoveComments” Ordner verwenden und bitte beachten Sie die Requirements, welche ich in der Anleitung dargestellt habe) erhält man die wie gehabt aus der Vorherigen Aufgabe die größten 3 Sünder. Jedoch wird der Nutzer ebenfalls gefragt, ob er seine Dateigrößen verkleinern will.

Ausgabe:

```
 MacBook Pro ~/Doc/schule/3Lehrjahr/M122 ➜ cat main +1 !1 ➜ sh duBytesFileSizeDecreasing.sh RemoveComments
Gesamtgröße: 1,19 KB

Die drei größten Dateien sind:

1. RemoveComments/removeComments.sh (622B)
2. RemoveComments/removeCommentTest.sh (490B)
3. RemoveComments/removeCommentsTest.js (110B)

Möchten Sie die Größe der Dateien verkleinern? (Ja/Nein) [
```

Im falle das er die Frage bestätigt erhält er diesen Output:

```
Möchten Sie die Größe der Dateien verkleinern? (Ja/Nein) ja

Entferne Kommentare und Leerzeichen aus den Dateien...

Neue Größe von 'RemoveComments/removeComments.sh': 480B
Neue Größe von 'RemoveComments/removeCommentTest.sh': 72B
Neue Größe von 'RemoveComments/removeCommentsTest.js': 66B

 MacBook Pro ~/Doc/schule/3Lehrjahr/M122 ➜ cat main +1 !2 ?2 [
```

Wobei ihm die neuen Größen der Dateien direkt wieder angezeigt werden. Hier ist noch ein Vorher nachher Beispiel, von einem dieser Files, die Kommentare von allen 3 aufgelisteten Files wurden entfernt:

```
1 # comments
2 # comments
3 # comments
4 # comments
5
6 # comments
7 echo "no comment"
8
9 # comments
10 # comments
11 # comments
12 # comments
13
14 # comments
15 # comments
16 # comments
17 # comments
18
19 # comments
20 # comments
21 # comments
22 # comments
23 # comments
24
25 # comments
26
27 echo "no comment"
28
29 echo "no comment"
30
31
32 echo "no comment"
33
34 # comments
35 |
```

```
RemoveComments > removeCommentTest.sh
1 echo "no comment"
2 echo "no comment"
3 echo "no comment"
4 echo "no comment"
5 |
```

Falls der Nutzer ablehnt ist dies die entsprechende Ausgabe:

```
Möchten Sie die Größe der Dateien verkleinern? (Ja/Nein) n
```

```
Die Größe der Dateien wird nicht verkleinert.
```

```
➔ ~/Documents/schule/3Lehrjahr/M122 ➔ main +1 !1 ?2 ➔
```

Aufgabe 4 Text processing

Anwenden von cat, uniq und sort

Um zwei Dateien zusammenzufügen, kann man den Befehl Join verwenden. Dabei muss man genau darauf achten, dass die korrekten Spaltennummern für den “join” Befehl verwendet werden.

Unset

```
join -1 3 -2 1 Person.txt Passwort.txt >  
UserAndPassword.txt
```

Ausgabe:

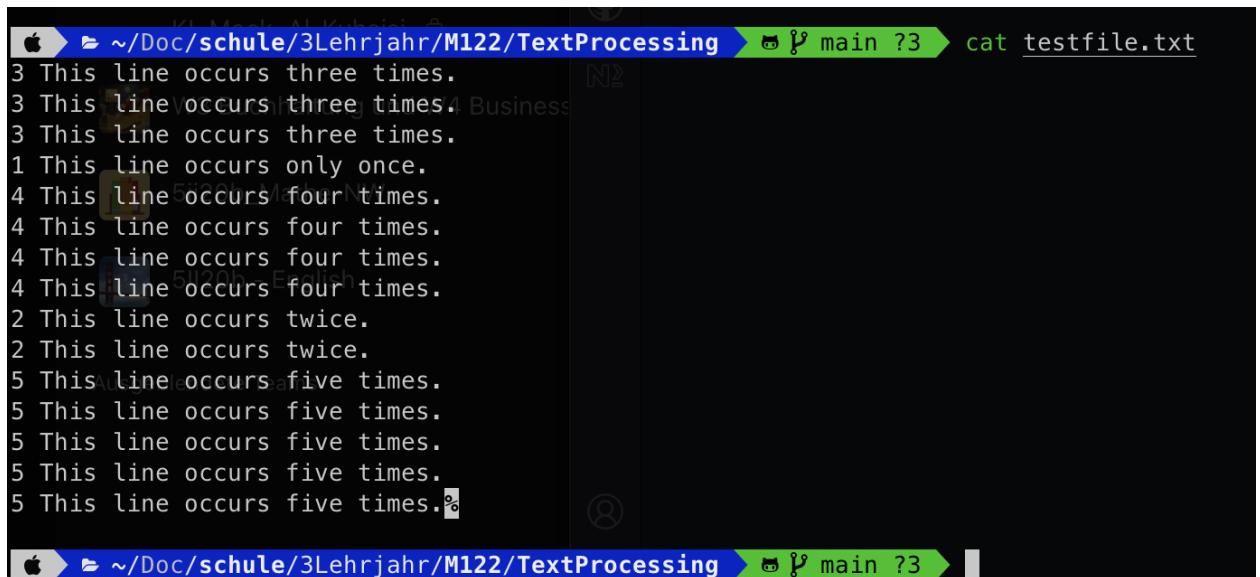
```
TextProcessing > UserAndPassword.txt  
1 1 Hans Muster user11 123@11  
2 2 Paul Keller user12 123@12  
3 3 Jose Pinto user13 123@13  
4 4 heinz Herrman user14 123@14  
5 5 Ruedi Falk user15 123@15  
6 |
```

Um den Inhalt einer Datei in einem Terminal auszugeben, kann der Befehl cat verwendet werden.

Unset

```
cat testfile.txt
```

Ausgabe:



```
3 This line occurs three times.  
3 This line occurs three times.  
3 This line occurs three times.  
1 This line occurs only once.  
4 This line occurs four times.  
2 This line occurs twice.  
2 This line occurs twice.  
5 This line occurs five times.  
5 This line occurs five times.%
```

Um doppelte Zeilen, welche sich in der Datei befinden, nur einmal anzuzeigen, kann man den Befehl "uniq" verwenden.

Unset

```
cat testfile.txt | uniq
```

Ausgabe:



```
3 This line occurs three times.  
1 This line occurs only once.  
4 This line occurs four times.  
2 This line occurs twice.  
5 This line occurs five times.%
```

Um den Output dann auch noch zu sortieren, kann man erneut mit einem Pipe auf den "sort" Befehl weiterleiten.

Unset

```
cat testfile.txt | uniq | sort
```

Ausgabe:

```
MacBook-Pro:~/Doc/schule/3Lehrjahr/M122/TextProcessing MacBook-Pro:~/Doc/schule/3Lehrjahr/M122/TextProcessing ➜ cat testfile.txt | uniq | sort
```

1 This line occurs only once.
2 This line occurs twice.
3 This line occurs three times.
4 This line occurs four times.
5 This line occurs five times.

```
MacBook-Pro:~/Doc/schule/3Lehrjahr/M122/TextProcessing MacBook-Pro:~/Doc/schule/3Lehrjahr/M122/TextProcessing ➜
```

Damit die entsprechende Sortierung dann auch noch numerisch sortiert wird, kann man die Option -n im Sortbefehl mitgeben.

Unset

```
cat testfile.txt | uniq | sort -n
```

Ausgabe:

```
MacBook-Pro:~/Doc/schule/3Lehrjahr/M122/TextProcessing MacBook-Pro:~/Doc/schule/3Lehrjahr/M122/TextProcessing ➜ cat testfile.txt | uniq | sort -n
```

1 This line occurs only once.
2 This line occurs twice.
3 This line occurs three times.
4 This line occurs four times.
5 This line occurs five times.

```
MacBook-Pro:~/Doc/schule/3Lehrjahr/M122/TextProcessing MacBook-Pro:~/Doc/schule/3Lehrjahr/M122/TextProcessing ➜
```

Für die Umgekehrte Sortierung kann dem “sort” Befehl ebenfalls noch ein -r für reverse, mitgegeben werden.

Unset

```
cat testfile.txt | uniq | sort -n -r
```

Ausgabe:

The terminal window shows the command being run and its output. The command is: `cat testfile.txt | uniq | sort -n -r`. The output is a list of numbers followed by text descriptions, sorted in descending order of frequency:

```
5 This line occurs five times.  
4 This line occurs four times.  
3 This line occurs three times.  
2 This line occurs twice.  
1 This line occurs only once.
```

Mehrere Dateien mit cat ausgeben

Github:

- Script https://github.com/lcmoc/M122_bash_scripts/blob/main/TextProcessing/textProcessing.sh

Um ein Skript zu schreiben, das den Inhalt der drei Dateien fox1.txt, fox2.txt und fox3.txt liest, sie sortiert, duplizierte Zeilen entfernt und das Ergebnis in final.txt speichert, muss man die folgenden Schritte ausführen:

1. cat-Befehl, um den Inhalt der drei Dateien in der Reihenfolge fox1.txt, fox2.txt und fox3.txt auszugeben.
2. Pipe-Symbol |, um den Output des cat-Befehls an den sort-Befehl weiterzugeben.
3. Pipe-Symbol |, um den Output des sort-Befehls an den uniq-Befehl weiterzugeben.
4. tee-Befehl, um das Ergebnis der Pipe in final.txt zu speichern.

Script:

```
1 #!/bin/bash
2
3 cat fox1.txt fox2.txt fox3.txt |
4     sort |
5     uniq |
6     tee final.txt
7 |
```

Ausgabe:

```
 MacBook Pro: ~/Doc/schule/3Lehrjahr/M122/TextProcessing ➜ ⌘ ⌘ main ➜ sh textProcessing.sh
The quick blue fox jumps over the lazy dog.
The quick blue frog jumps over the lazy dog.
The quick blue mice jumps over the lazy dog.
The quick brown cat jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.
The quick darkyellow fox jumps over the lazy dog.
The quick green cat jumps over the lazy dog.
The quick green fox jumps over the lazy dog.
The quick green mice jumps over the lazy dog.
The quick silver fox jumps over the lazy dog.
The quick white cat jumps over the lazy dog.
The quick white fox jumps over the lazy dog.
The quick yellow dog jumps over the lazy dog.
The quick yellow fox jumps over the lazy dog.
The quick yellow frog jumps over the lazy dog.
```

```
 MacBook Pro: ~/Doc/schule/3Lehrjahr/M122/TextProcessing ➜ ⌘ ⌘ main ?2 ➜
```

```
TextProcessing > final.txt
1 The quick blue fox jumps over the lazy dog.
2 The quick blue frog jumps over the lazy dog.
3 The quick blue mice jumps over the lazy dog.
4 The quick brown cat jumps over the lazy dog.
5 The quick brown fox jumps over the lazy dog.
6 The quick darkyellow fox jumps over the lazy dog.
7 The quick green cat jumps over the lazy dog.
8 The quick green fox jumps over the lazy dog.
9 The quick green mice jumps over the lazy dog.
10 The quick silver fox jumps over the lazy dog.
11 The quick white cat jumps over the lazy dog.
12 The quick white fox jumps over the lazy dog.
13 The quick yellow dog jumps over the lazy dog.
14 The quick yellow fox jumps over the lazy dog.
15 The quick yellow frog jumps over the lazy dog.
16
```

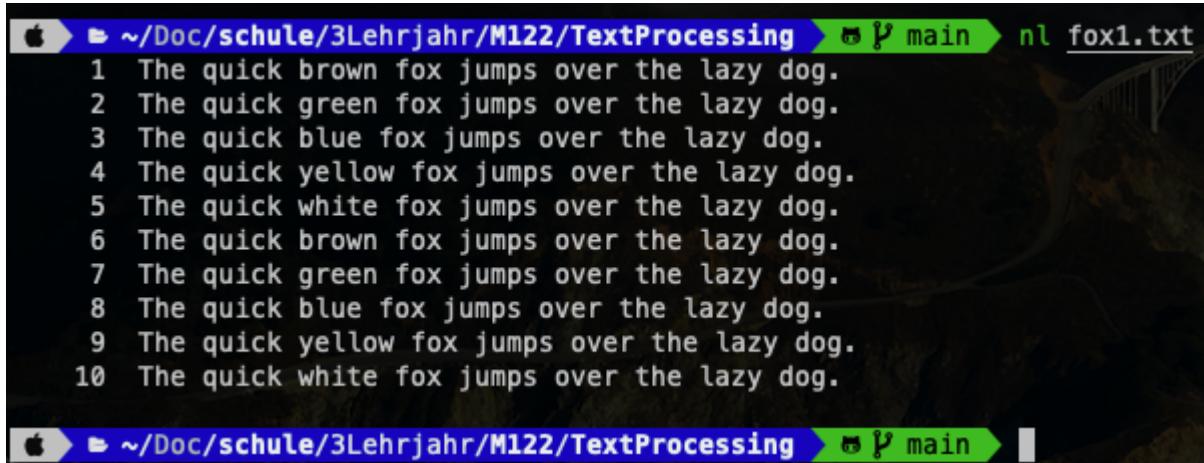
Eine Datei mit Zeilennummern versehen

Um eine Datei mit Zeilennummern zu versehen, muss der Befehl "nl" mit dem Filenamen angegeben werden.

```
Unset
```

```
nl fox1.txt
```

Ausgabe:



The screenshot shows two terminal sessions. The top session shows the command `nl fox1.txt` being run, and the bottom session shows the resulting numbered list of text lines.

```
~$ ~/Doc/schule/3Lehrjahr/M122/TextProcessing > nl fox1.txt
1 The quick brown fox jumps over the lazy dog.
2 The quick green fox jumps over the lazy dog.
3 The quick blue fox jumps over the lazy dog.
4 The quick yellow fox jumps over the lazy dog.
5 The quick white fox jumps over the lazy dog.
6 The quick brown fox jumps over the lazy dog.
7 The quick green fox jumps over the lazy dog.
8 The quick blue fox jumps over the lazy dog.
9 The quick yellow fox jumps over the lazy dog.
10 The quick white fox jumps over the lazy dog.
```

Statt Seitenweise Man-Page Nur Das Wichtigste

Bei diesem Script geht es darum, die Ausgabe vom "man" Befehl zu verkleinern und eine kurze Beschreibung auszugeben.

Github:

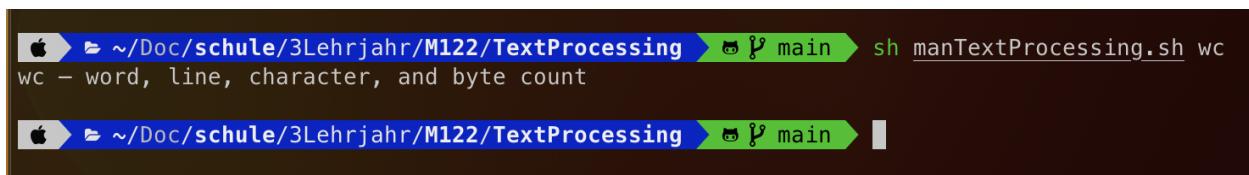
- Script

https://github.com/lcmoc/M122_bash_scripts/blob/main/TextProcessing/manTextProcessing.sh

Script:

```
1  #!/bin/bash
2
3  if [ $# -eq 0 ]; then
4      echo "Kein Befehl angegeben. Bitte geben Sie einen Befehl als Parameter an."
5      exit 1
6  fi
7
8  command=$1
9
10 man "$command" | head -n4 | tail -n1 | cut -b 8-
11 
```

Ausgabe:



The terminal window shows the following session:

```
MacBook-Pro:~/Doc/schule/3Lehrjahr/M122/TextProcessing lcmoc$ sh manTextProcessing.sh wc
wc - word, line, character, and byte count

MacBook-Pro:~/Doc/schule/3Lehrjahr/M122/TextProcessing lcmoc$ 
```

Aufgabe 5 Umgestalten

Schritt 1 „Prüfen von Input und Output Parameter“

Github:

- Script

https://github.com/lcmoc/M122_bash_scripts/blob/main/Umgestalten/convertData01_Parameter.sh

Bei diesem Schritt wird ein Script erstellt, welches überprüft, ob eine Datei als Parameter hinzugefügt wurde.

Script:

```
You, 4 hours ago | 1 author (You)
1 #!/bin/bash
2
3 input="$1" You, 4 hours ago • finished script 4
4 output="$2"
5
6 if [ -z $input ] || [ -z $output ]
7 then
8     echo "Dieses Script filtert aus einer Speziellen Liste"
9     echo "die deutschsprachigen Hauptarbeitsbezeichnungen heraus."
10    echo ""
11    echo "Sie müssen beim Aufruf des Programms $0 den namen der Input-,"
12    echo "und der Outpit-Datei mitgeben."
13    exit 1
14 fi
15
16 exit 0
17
```

Ausgabe im Fehlerfall:

```
~/Doc/schule/3Lehrjahr/M122/Umgestalten ➤ sh convertData01_Parameter.sh
Dieses Script filtert aus einer Speziellen Liste
die deutschsprachigen Hauptarbeitsbezeichnungen heraus.

Sie müssen beim Aufruf des Programms convertData01_Parameter.sh den namen der Input-,
und der Outpit-Datei mitgeben.
```

Schritt 2 „Alle deutschsprachigen Texte herausfiltern“

Github:

- Script:
https://github.com/lcmoc/M122_bash_scripts/blob/main/Umgestalten/convertD ata02_Parameter.sh
- Ausgabedatei:
https://github.com/lcmoc/M122_bash_scripts/blob/main/Umgestalten/output02 .txt

Hier sollen (Aufbauend auf das vorherige Script) alle Deutschen Texte aus einer "spl" Datei heraussortiert und in einer neuen "txt" Datei angezeigt werden. Dies ist möglich, da die Zeilen, welche auf Deutsch sind, mit einer 2 markiert sind.

Die Datei ist wie folgt aufgebaut:

Unset

```
2 5 'M041010' 'Dichtringe der Hydraulikleitungen am Lenkgetriebe ersetzen'
```

Durch den "grep" Befehl ist es möglich alle Spalten welche eine 2 haben, herauszufiltern

Unset

```
grep '^2' "$input"
```

Script:

```
You, 5 hours ago | 1 author (You)
1 input="$1"
2 output="$2"
3
4 if [ -z $input ] || [ -z $output ]
5 then
6     echo "Dieses Script filtert aus einer Speziellen Liste"
7     echo "die deutschsprachigen Hauptarbeitsbezeichnungen heraus."
8     echo " "
9     echo "Sie müssen beim Aufruf des Programms $0 den namen der Input-,"
10    echo "und der Outpit-Datei mitgeben."
11    exit 1
12 fi
13
14 grep -a -w "^\?" $input > $output
15
16 exit 0
17
```

Ausgabe:

```
di einzig schwarz
/Applications/PyCharm.app/Contents/bin/python main.py
sh convertData02_Parameter.sh Export.spl output02.txt
/Applications/PyCharm.app/Contents/bin/python main.py
mathias brüder vater
```

115997	2	24	'B005020107'	'Schmutzfänger vorn aus- und einbauen'
115998	2	24	'B005020108'	'Schmutzfänger hinten aus- und einbauen'
115999	2	24	'B005020116'	'Seitenfenster der A- Säule aus- und einbauen'
116000	2	24	'B00502017'	'Wagen außen - Zweischicht-Lackierung - Hauptvorgaben'
116001	2	24	'B005129'	'Wagenseite komplett bis Fensterlinie einschließlich Innenlackierung <Rechts>'
116002	2	24	'B00512900'	'Wagenseite komplett bis Fensterlinie einschließlich Innenlackierung <Rechts> - Einschicht-Lackierung - Hauptvorgaben'
116003	2	24	'B00512917'	'Wagenseite komplett bis Fensterlinie einschließlich Innenlackierung <Rechts> - Zweischicht-Lackierung - Hauptvorgaben'
116004	2	24	'B005130'	'Wagenseite komplett bis Dachanschluß einschließlich Innenlackierung <Rechts>'
116005	2	24	'B00513000'	'Wagenseite komplett bis Dachanschluß einschließlich Innenlackierung <Rechts> - Einschicht-Lackierung - Hauptvorgaben'
116006	2	24	'B00513017'	'Wagenseite komplett bis Dachanschluß einschließlich Innenlackierung <Rechts> - Zweischicht-Lackierung - Hauptvorgaben'
116007	2	24	'B005131'	'Wagenseite oberhalb der Zierleiste bis Fensterlinie <Rechts>'
116008	2	24	'B00513100'	'Wagenseite oberhalb der Zierleiste bis Fensterlinie <Rechts> - Einschicht-Lackierung - Hauptvorgaben'
116009	2	24	'B005131040'	'Gesamt-Vorgabe'
116010	2	24	'B005131046'	'Scheinwerfer aus- und einbauen'
116011	2	24	'B005131049'	'Heckleuchten aus- und einbauen'

Schritt 3 Alle Zeilen mit einer „26“ herausfiltern

Github:

- Script:

https://github.com/lcmoc/M122_bash_scripts/blob/main/Umgestalten/convertD ata03_alleOhne26.sh

- Ausgabedatei:

https://github.com/lcmoc/M122_bash_scripts/blob/main/Umgestalten/output03 .txt

Dieses Script ist auch in diesem Fall auf das vorherige Script (Schritt 2) aufgebaut.

Bei diesem Script geht es darum, dass die 26 aus der zweiten Zeile entfernt wird, dies konnte ich ebenfalls mit dem befehl grep erreichen.

Unset

```
grep -a -v "26"
```

Script:

```
You, 6 hours ago | 1 author (You)
#!/bin/bash

input="$1"
output="$2"

if [ -z $input ] || [ -z $output ]
then
    echo "Dieses Script filtert aus einer Speziellen Liste"
    echo "die deutschprachigen Hauptarbeitsbezeichnungen heraus."
    echo ""
    echo "Sie müssen beim Aufruf des Programms $0 den namen der Input-,"
    echo "und der Outpit-Datei mitgeben."
    exit 1
fi

grep -a -v "26" $input | grep -a -w "^\?" > $output

exit 0
```

You, 6 hours ago • finished script 4

Ausgabe:

```
➔ ~/Doc/schule/3Lehrjahr/M122/Umgestalten ➔ ⚡ main !1 ➔ sh convertData03_alleOhne26.sh Export.spl output03.txt
➔ ~/Doc/schule/3Lehrjahr/M122/Umgestalten ➔ ⚡ main !1 ➔
```

28181	2	5	'J314500010'	'Zwei Ventilstößel/Hydrostößel ersetzen'
28182	2	5	'J314500015'	'Mehrarbeit bei Fahrzeugen mit Motorraumabdeckung unten'
28183	2	5	'J314500019'	'Einen weiteren oder alle Ventilstößel/Hydrostößel ersetzen'
28184	2	5	'J314500020'	'Alle Ventilstößel/Hydrostößel eines anderen Zylinders ersetzen'
28185	2	5	'J314500021'	'Alle Ventilstößel/Hydrostößel ersetzen'
28186	2	5	'J314550'	'Nockenwelle (Einlaß) aus- und einbauen'
28187	2	5	'J314550002'	'Mehrarbeit bei Fahrzeugen mit Motorraumabdeckung unten'
28188	2	5	'J314550004'	'Einen Ventilstößel/Hydrostößel ersetzen'
28189	2	5	'J314550005'	'Einen weiteren oder alle Ventilstößel/Hydrostößel ersetzen'
28190	2	5	'J314560'	'Nockenwelle (Auslaß) aus- und einbauen'
28191	2	5	'J314560002'	'Mehrarbeit bei Fahrzeugen mit Motorraumabdeckung unten'
28192	2	5	'J314560004'	'Einen Ventilstößel/Hydrostößel ersetzen'
28193	2	5	'J314560005'	'Einen weiteren oder alle Ventilstößel/Hydrostößel ersetzen'

Schritt 4 „Spalte 1 und 2 sollen verschwinden“

Github:

- Script:
https://github.com/lcmoc/M122_bash_scripts/blob/main/Umgestalten/convertData04_SpaltenWeg.sh
- Ausgabedatei:
https://github.com/lcmoc/M122_bash_scripts/blob/main/Umgestalten/output04.txt

Dieses Script baut ebenfalls auf die vorherige Aufgabe auf.

Hierbei sollen die ersten beiden Zeilen aus der Datei entfernt und in einer neu generierten Textdatei, welche als Parameter übergeben wird, eingefügt werden, dies habe ich mit dem “cut” Befehl erreicht.

Unset

```
cut -f3-
```

Script:

```

...
1 #!/bin/bash|      You, 6 hours ago • finished script 4 ...
2 input="$1"
3 output="$2"
4
5 if [ -z $input ] || [ -z $output ]
6 then
7     echo "Dieses Script filtert aus einer Speziellen Liste"
8     echo "die deutschprachigen Hauptarbeitsbezeichnungen heraus."
9     echo ""
10    echo "Sie müssen beim Aufruf des Programms $0 den namen der Input-,"
11    echo "und der Outpit-Datei mitgeben."
12    exit 1
13 fi
14
15 grep '^2' "$input" | grep -a -v "26" | cut -f3- > "$output"
16

```

Ausgabe:

```

❯ ~/Doc/schule/3Lehrjahr/M122/Umgestalten ➔ ⌂ P main !1 ➔ sh convertData04_SpaltenWeg.sh Export.spl output04.txt
❯ ~/Doc/schule/3Lehrjahr/M122/Umgestalten ➔ ⌂ P main !1 ➔

```

```

Umgestalten > 📄 output04.txt
You, 6 hours ago | 1 author (You)
1 'A001500'      'Teilersatz Karosserieseitenwand vorn – Rechte Seite'
2 'Schutzzierleiste an Seitenwand vorn anbringen'
3 'A001500005'   'B-Säule oberhalb ersetzen'
4 'A001600'      'Teilersatz Karosserieseitenwand vorn – Linke Seite'
5 'A001600001'   'Schutzzierleiste an Seitenwand vorn anbringen'
6 'A001600005'   'B-Säule oberhalb ersetzen'
7 'A201598'      'Bodenblech vorn abdichten'
8 'A203098'      'Bodenblech mitten abdichten'
9 'A205098'      'Bodenblech hinten abdichten'

```

Schritt 5 „Nur die Hauptgruppen ausgeben“

Github:

- Script:

https://github.com/lcmoc/M122_bash_scripts/blob/main/Umgestalten/convertD ata05_NurHauptgruppen.sh

- Ausgabedatei:

https://github.com/lcmoc/M122_bash_scripts/blob/main/Umgestalten/convertD ata05_NurHauptgruppen.sh

Bei diesem Script geht es darum, dass nur die Hauptgruppen in eine neue Datei geschrieben werden. Das heisst nur die Zeilen, welche mit einem Buchstaben von A-Z anfangen und mit 6 folgenden Ziffern endet. Das Script soll ebenfalls auf die vorherigen Scripts aufbauen.

Um dies zu erreichen, habe ich einen Regex in den Befehl eingebaut, welcher das obene Pattern genau filtert.

Unset

```
grep -Eo '[A-Z][0-9]{6}\b.*'
```

Script:

```

You, 8 hours ago | 1 author (You)
1 #!/bin/bash
2
3 input="$1"
4 output="$2"
5
6 if [ -z $input ] || [ -z $output ]
7 then
8     echo "Dieses Script filtert aus einer Speziellen Liste"
9     echo "die deutschsprachigen Hauptarbeitsbezeichnungen heraus."
10    echo ""
11    echo "Sie müssen beim Aufruf des Programms $0 den namen der Input-,"
12    echo "und der Output-Datei mitgeben."
13    exit 1
14 fi
15
16 grep '^2' "$input" | grep -a -v "26" | cut -f3- | grep -Eo '[A-Z][0-9]{6}\b.*' > "$output"
17

```

Ausgabe:

```

You, 8 hours ago | 1 author (You)
1 A001500' 'Teilersatz Karosserieseitenwand vorn - Rechte Seite'
2 A001600' 'Teilersatz Karosserieseitenwand vorn - Linke Seite'
3 A201598' 'Bodenblech vorn abdichten'
4 A203098' 'Bodenblech mitten abdichten'
5 A205098' 'Bodenblech hinten abdichten'
6 A206098' 'Stirnwand abdichten'
7 A210198' 'Regenrinne abdichten - Rechte Seite'
8 A210298' 'Regenrinne abdichten - Linke Seite'
9 A213599' 'Windlauf instandsetzen'
10 A215598' 'Karosserierückwand unten abdichten'
11 A215599' 'Karosserierückwand unten instandsetzen'
12 A216398' 'Radeinbau vorn abdichten - Rechte Seite'
13 A216498' 'Radeinbau vorn abdichten - Linke Seite'
14 A216598' 'Radeinbau hinten abdichten - Rechte Seite'
15 A216698' 'Radeinbau hinten abdichten - Linke Seite'
16 A217198' 'Karosserieseitenwand abdichten - Rechte Seite'
17 A217298' 'Karosserieseitenwand abdichten - Linke Seite'
18 A217598' 'Einstiegverkleidung abdichten - Rechte Seite'
19 A217698' 'Einstiegverkleidung abdichten - Linke Seite'
20 A218198' 'Kotflügel abdichten - Rechte Seite'
21 A218298' 'Kotflügel abdichten - Linke Seite'

```

Aufgabe 6 Funktionen

Zusammenzählen

Github:

- Script:

https://github.com/lcmoc/M122_bash_scripts/blob/main/Funktionen/zusammenzaelen.sh

Es soll ein Script erstellt werden, welches 2 Zahlen als Inputs nimmt und diese dann addiert, für die Addition soll in dem Fall eine Funktion verwendet werden.

Script:

```
You, 5 minutes ago | 1 author (You)
1 #!/bin/bash
2
3 zaehleVonBis() {
4     start=$1
5     ende=$2
6     summe=0
7     for (( zahl=start; zahl<=ende; zahl++ ))
8     do
9         (( summe+=zahl ))
10    done
11    echo $summe
12 }
13
14 read -p "Gib die Startzahl ein: " start
15 read -p "Gib die Endzahl ein: " ende
16
17 ergebnis=$(zaehleVonBis $start $ende)
18
19 echo "Die Summe von $start bis $ende ist $ergebnis."
20
```

Ausgabe:

```
mac ~Documents/schule/3Lehrjahr/M122/Funktionen ➔ main ➔ sh zusammenzaelen.sh
Gib die Startzahl ein: 2
Gib die Endzahl ein: 2
Die Summe von 2 bis 2 ist 2.

mac ~Doc/schule/3Lehrjahr/M122/Funktionen ➔ main ➔
```

Zusatz "Rechner"

Github:

- Script:

https://github.com/lcmoc/M122_bash_scripts/blob/main/Funktionen/rechner.sh

Bei dieser Zusatzaufgabe habe ich einen Rechner erstellt, der über eine Funktion Multiplikation, Division, Addition und Subtraktion Aufgaben erfüllen kann.

Script:

```
You, 9 minutes ago | 1 author (You)
1 #!/bin/bash
2
3 rechnen() {
4     operation=$1
5     zahl1=$2
6     zahl2=$3
7     case $operation in
8         +)
9             echo $((zahl1 + zahl2))
10            ;;
11        -)
12            echo $((zahl1 - zahl2))
13            ;;
14        \x)
15            echo $((zahl1 * zahl2))
16            ;;
17        /)
18            echo $(bc -l <<< "scale=2; $zahl1 / $zahl2")
19            ;;
20        *)
21            echo "Ungültige Operation!"
22            exit 1
23            ;;
24    esac
25 }
26
27 read -p "Gib die erste Zahl ein: " zahl1
28 read -p "Gib die zweite Zahl ein: " zahl2
29 read -p "Welche Operation möchtest du durchführen (+, -, x, /)? " operation
30
31 ergebnis=$(rechnen $operation $zahl1 $zahl2)
32
33 echo "$zahl1 $operation $zahl2 = $ergebnis"
34 |
```

Ausgabe:

```
 MacBook-Pro: ~/Documents/schule/3Lehrjahr/M122/Funktionen ➜ ⌂ main ➜ sh rechner.sh
Gib die erste Zahl ein: 2
Gib die zweite Zahl ein: 2
Welche Operation möchtest du durchführen (+, -, x, /)? +
2 + 2 = 4

 MacBook-Pro: ~/Doc/schule/3Lehrjahr/M122/Funktionen ➜ ⌂ main ➜ sh rechner.sh
Gib die erste Zahl ein: 2
Gib die zweite Zahl ein: 2
Welche Operation möchtest du durchführen (+, -, x, /)? -
2 - 2 = 0

 MacBook-Pro: ~/Doc/schule/3Lehrjahr/M122/Funktionen ➜ ⌂ main ➜ sh rechner.sh
Gib die erste Zahl ein: 2
Gib die zweite Zahl ein: 2
Welche Operation möchtest du durchführen (+, -, x, /)? x
2 x 2 = 4

 MacBook-Pro: ~/Doc/schule/3Lehrjahr/M122/Funktionen ➜ ⌂ main ➜ sh rechner.sh
Gib die erste Zahl ein: 2
Gib die zweite Zahl ein: 2
Welche Operation möchtest du durchführen (+, -, x, /)? /
2 / 2 = 1.00

 MacBook-Pro: ~/Doc/schule/3Lehrjahr/M122/Funktionen ➜ ⌂ main ➜ sh rechner.sh
Gib die erste Zahl ein: 2
Gib die zweite Zahl ein: 2
Welche Operation möchtest du durchführen (+, -, x, /)? f
2 f 2 = Ungültige Operation!

 MacBook-Pro: ~/Doc/schule/3Lehrjahr/M122/Funktionen ➜ ⌂ main ➜
```

Befehlsübersicht

Die komplette Befehlsübersicht ist auf einem Git Repository ersichtlich

https://github.com/lcmoc/M122_bash_scripts/blob/main/befehlsuebersicht.md

Befehl	Erklärung
while	Der while-Befehl wird verwendet, um eine Schleife zu erstellen, die wiederholt ausgeführt wird, solange eine bestimmte Bedingung erfüllt ist.
do	Der do-Befehl wird in Verbindung mit dem while-Befehl verwendet und gibt an, welche Aktionen innerhalb der Schleife ausgeführt werden sollen.
if	Der if-Befehl wird verwendet, um eine Bedingung zu testen und abhängig vom Ergebnis verschiedene Aktionen auszuführen.
=~	Der =~-Befehl wird verwendet, um eine reguläre Ausdruck Übereinstimmung in einer Bedingung zu testen.
read	Der read-Befehl wird verwendet, um Benutzereingaben in eine Variable zu speichern.
continue	Der continue-Befehl wird innerhalb einer Schleife verwendet, um die aktuelle Iteration zu beenden und zur nächsten Iteration fortzufahren.
echo	Der echo-Befehl wird verwendet, um Text auf dem Bildschirm auszugeben.
then	Der then-Befehl wird in Verbindung mit dem if-Befehl verwendet und gibt an, welche Aktionen ausgeführt werden sollen, wenn die Bedingung wahr ist.
fi	Der fi-Befehl beendet einen if-Block.
break	Der break-Befehl wird innerhalb einer Schleife verwendet, um die Schleife vorzeitig zu beenden.
^[0-9]+\$	Dies ist ein regulärer Ausdruck, der auf eine Zeichenkette passt, die nur aus Ziffern besteht.
RANDOM	Die Variable RANDOM enthält eine zufällige Ganzzahl.

%	Der Modulo-Operator % wird verwendet, um den Rest einer Division zu berechnen.
+	Der Plus-Operator + wird verwendet, um zwei Zahlen zu addieren.
-lt	Der Vergleichsoperator -lt wird verwendet, um zu testen, ob eine Zahl kleiner als eine andere Zahl ist.
-gt	Größer als check >
ls	Listet den Inhalt eines Verzeichnisses auf.
rmdir	Entfernt ein leeres Verzeichnis.
cat	Ein Befehl, der verwendet wird, um den Inhalt von Dateien anzuzeigen oder zu kombinieren.
du	Gibt die Größe von Dateien und Verzeichnissen auf der Festplatte aus.
--apparent-size:	Gibt die scheinbare Größe einer Datei aus.
-s	Gibt die Gesamtgröße eines Verzeichnisses an.
-h	Gibt die Größenangabe in einer für Menschen lesbaren Form aus.