

# bash Workshop

Markus Wust und Martin Raden

Universitätsbibliothek und Dr. Eberle Zentrum für digitale Kompetenzen

Universität Tübingen

Folien unter <https://t1p.de/d12dq>

# Was soll rauskommen

Sie haben eine Idee ...

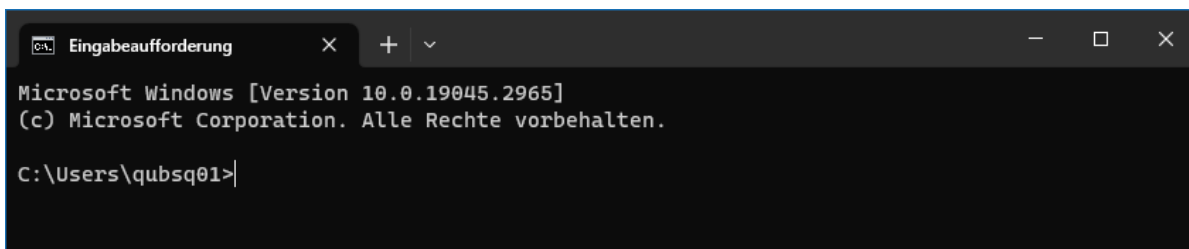
... was eine Shell ist,  
... was man damit tun *könnte*, und  
... wie sie grundlegende Schritte  
umsetzen können.

Sie haben aktiv ...

... erste Schritte in der Shell getan,  
... Wildcards verstanden, und  
... Lust auf mehr!

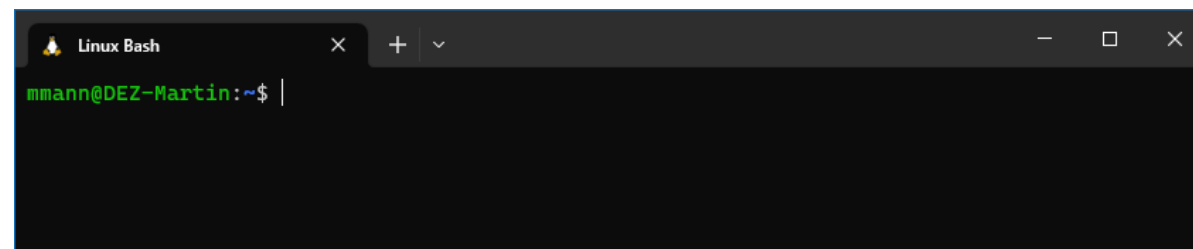
# Die Shell – Was ist das?

# MS DOS / PowerShell vs. Linux Shell (Bash, zsh, ...)



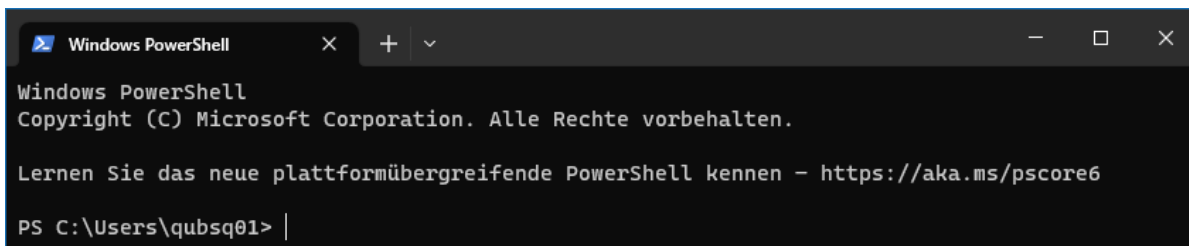
```
Eingabeaufforderung
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\qubsq01>
```



```
Linux Bash
mmann@DEZ-Martin:~$
```

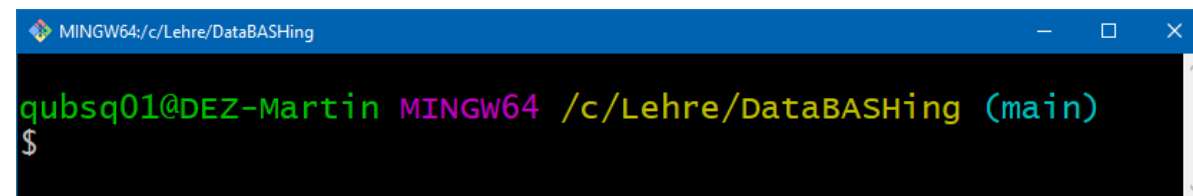
- MS DOS = Eingabeaufforderung
- MS PowerShell = erweiterter Nachfolger mit mehr Funktionalität
- Unix / Linux / macOS Shell
  - Bash = weit verbreitet
  - zsh = umfangreich
- via MINGW, Cygwin, MS WSL, ... auch in Windows verfügbar



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

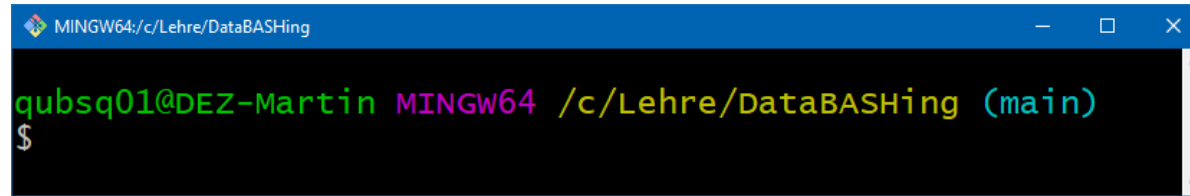
Lernen Sie das neue plattformübergreifende PowerShell kennen - https://aka.ms/pscore6

PS C:\Users\qubsq01>
```



```
MINGW64:/c/Lehre/DataBASHing
qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBASHing (main)
$
```

# Was ist das?



```
MINGW64:/c/Lehre/DataBASHing
qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBASHing (main)
$
```

- Git Bash = **abgespecktes** Linux (MINGW64) als Programm in MS Windows
- **Benutzername @ Computernamen**
- **Aktuelles Verzeichnis** in Linux Notation ("/" statt "\" als Trennzeichen)
- **Git branch** (hier irrelevant)

**\$** = Eingabeprompt wartet auf Eingabe von uns

Empfehlung: ggf. besser [WSL \(Windows Subsystem for Linux\)](#) installieren = komplettes Linux unter Windows

Und was macht man damit?

# Anwendungsbeispiel

[Image by Damian Zaleski](#)



# Navigation in der Kommandozeile

ls, cd, pwd, cat, head, tail



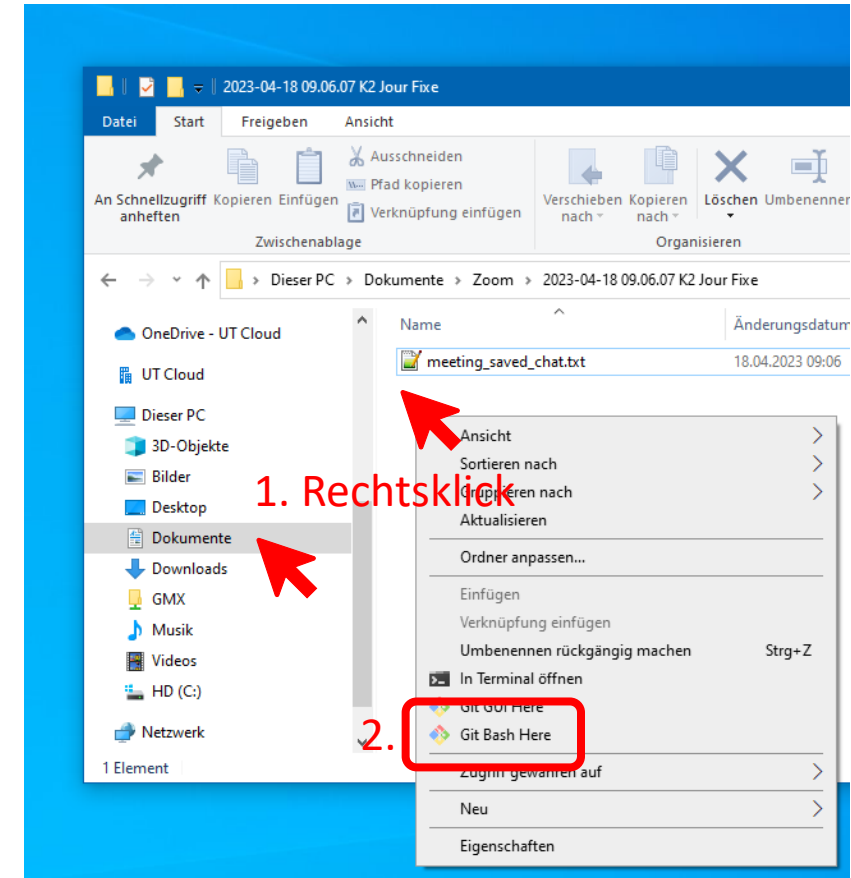
# Wie komme ich da hin?

- **Kontextmenü** im Explorer (siehe rechts)
  - Rechtsklick auf Ordner oder Hintergrund
  - „**Git Bash Here**“ im Menü auswählen

oder

- **Git Bash Terminal** (irgendwo/-wie) **öffnen**
  - `$ cd "C:\Daten\Sonstewo"` = Windows-artiger Pfad oder
  - `$ cd /c/Daten/Sonstewo` = Linux-artiger Pfad
  - in Hochkommas setzen, wenn Leer- oder Sonderzeichen im Pfad

- mit `$ pwd` (parent working directory) kann man sehen, wo man gelandet ist ...



# Und nun?

**\$ cd** = change directory

- Zielort z.B. Unterverzeichnis oder Pfad

**\$ cd ..** = übergeordneter Ordner

```
MINGW64/c/Lehre/DataBASHing
qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBASHing (main)
$ cd data

qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBASHing/data (main)
$ ls
elements.zip  planets.txt  salmon.txt  sunspot.txt

qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBASHing/data (main)
$ ls -l *.txt
-rw-r--r-- 1 qubsq01 197121  8898 Jul 30  2021 planets.txt
-rw-r--r-- 1 qubsq01 197121    45 Jul 30  2021 salmon.txt
-rw-r--r-- 1 qubsq01 197121 73861 Jul 30  2021 sunspot.txt

qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBASHing/data (main)
$ cd ..

qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBASHing (main)
$
```

**\$ ls** = list (Inhalt = Dateien + Verzeichnisse)

- ggf. Namensmuster mit Platzhaltern für beliebige Zeichen **\*** oder *ein* Zeichen **?**

Allgemeiner  
Aufbau von  
Aufrufen:

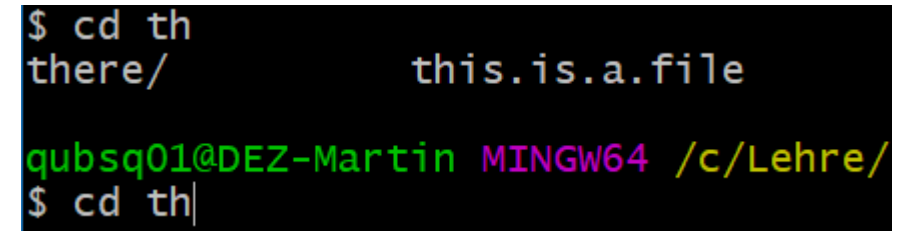
PROGRAMM ARGUMENTE ZIELNAME			
• ein Wort	Leerzeichen	• optional • starten mit „-“ • mehrere mittels <i>Leerzeichen</i> <i>getrennt</i> möglich	• z.B. Dateiname oder Pfad (relativ oder absolut) • kann <i>Platzhalter</i> („*“ und „?“) enthalten • in <i>Hochkommas</i> , wenn Leerzeichen enthalten! • <i>Autovervollständigung</i> via „TAB“ Taste



# Auto-completion = Magic !

- **Tabulatortaste** (ggf. 2x drücken) bei unvollständiger Eingabe von

- Programmnamen
- Dateinamen
- Verzeichnisnamen



```
$ cd th  
there/          this.is.a.file  
  
qubsq01@DEZ-Martin MINGW64 /c/Lehre/  
$ cd th|
```

liefert **Vorschlagsliste** (oder ergänzt einzige Möglichkeit)

- **Beschleunigt Navigation und Kommandobau enorm !!!**

# „Einsichten mit der Katze“

**\$ cat DATEI** = conCAtenate

- Kombinierte *vollständige Ausgabe* der Zieldateien

**\$ head DATEI** = nur erste KOPFzeilen

**-n ANZAHL** = Ausgabelänge

**\$ tail DATEI** = nur letzte Zeilen (= Schwanzende)

**\$ PROGRAM --help** liefert i.d.R. Übersicht der Möglichkeiten

# Na dann mal los !!!

- Hands-on Übung „Navigation in der Kommandozeile“



[Image by Damian Zaleski](#)

# Datei- und Verzeichnismanagement

cp, mv, rm, mkdir, wildcards \*

# Kopieren von Dateien

**\$ cp DATEI ZIEL** = **copy** einer DATEI erstellen

- **ZIEL** = *Dateiname* ODER *Pfad* (Name wird beibehalten) ODER *Pfad/Dateiname*

**\$ cp D1 D2 D3 ZIELPFAD** = **mehrere Dateien** kopieren

- **ZIELPFAD** = *immer* letztes Argument (muss hier ein Pfad sein!)

**\$ cp \*.txt ZIELPFAD** = Dateiauswahl via **WILDCARD** Pattern

**\*** = Platzhalter für **beliebig viele beliebige** Buchstaben (ausser Leerzeichen)

**?** = Platzhalter für **EINEN beliebigen** Buchstaben

**ACHTUNG: Unterschied zu RegEx!**



# Sind sie fit? Wildcard Frage 1

- **Grundidee**: finden sie Gemeinsamkeiten der gesuchten Dateien und ersetzen sie Unterschiede durch die Wildcards **\*** und **?**

```
createbackup.sh  list.sh  lspace.sh      speaker.sh  
listopen.sh     lost.sh  rename-files.sh  topprocs.sh
```

1\*

Umfasst wieviele Dateien?

- A. 1
- B. 2
- C. 3
- D. 4



# Sind sie fit? Wildcard Frage 2

- **Grundidee**: finden sie Gemeinsamkeiten der gesuchten Dateien und ersetzen sie Unterschiede durch die Wildcards **\*** und **?**

```
createbackup.sh  list.sh  lspace.sh  speaker.sh  
listopen.sh     lost.sh  rename-files.sh  topprocs.sh
```

??s\*

Umfasst wieviele Dateien?

- A. 1
- B. 2
- C. 3
- D. 4

# Sind sie fit? Wildcard Frage 3

- Next Level: Mit **[ ]** können **Buchstabengruppen** definiert werden, z.B. **[xyz]** oder **[0-9]** oder **[a-zA-Z]** oder sogar negiert **[!abc]**

```
users-111.list  users-1AA.list  users-22A.list  users-2aB.txt
users-111.txt   users-1AA.txt   users-22A.txt   users-2AB.txt
users-11A.txt   users-1AB.list  users-2aA.txt   users-2ba.list
```

```
users-[0-9][a-zA-Z][0-9]*
```

Umfasst wieviele?

- A. 1
- B. 2
- C. 3
- D. 4

# Kopieren von Verzeichnissen

```
$ cp -r VERZEICHNIS ZIELPFAD
```

- kopiert das Verzeichnis samt Inhalt in den ZIELPFAD

**-r** = **rekursiv** = auch alle **Dateien und Unterordner** kopiert

**ZIELPFAD** muss mit **/** oder **/. enden**

- **Absolute Pfade** *beginnen* mit **/**
- **Relative Pfade** (in Relation zum aktuellen Verzeichnis) *beginnen* mit
  - **.** = aktuelles Verzeichnis
  - **..** = übergeordnetes Verzeichnis
  - **ORDNERNAME** im aktuellen Verzeichnis

```
qubsq01@DEZ-Martin MINGW64 /c/Lehre/Dat
$ ls
here/  maybe/  or/  there/  this.is.a.f

qubsq01@DEZ-Martin MINGW64 /c/Lehre/Dat
$ cp -r here maybe/

qubsq01@DEZ-Martin MINGW64 /c/Lehre/Dat
$ ls maybe/
here/
```

# Umbenennen und Verschieben

```
$ mv X Y Z ZIELPFAD
```

= **move** von mehreren Dingen in **ZIELPFAD**

- **Wildcard Pattern** möglich
- **ZIELPFAD** muss mit **/** oder **/.** enden

```
$ mv ALT NEU
```

 = umbenennen  
*einer* Datei oder *eines* Verzeichnisses

- technisch wie verschieben,  
daher nur ein Befehl

```
qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBAS
$ ls
here/  maybe/  or/  there/  this.is.a.file

qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBAS
$ mv this.is.a.file still.a.file

qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBAS
$ ls
here/  maybe/  or/  still.a.file  there/

qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBAS
$ mv still.a.file here/

qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBAS
$ ls
here/  maybe/  or/  there/
```

# Löschen und Anlegen

```
$ rm x.xml *.txt /tmp/doof.csv
```

 = remove Dateien

- Wildcard Pattern, absolute + relative Pfade, ... alles verwendbar

```
$ rm -r ./here/ /tmp/gedoens/
```

 = remove Verzeichnis

- **-r** = rekursiv = alle enthaltenen Dateien und Unterordner !

- **ACHTUNG:** **UNWIEDERBRINGLICH !!!** Kein Papierkorb!

```
$ mkdir ORDNER
```

 = make directory

# Automatisierung

loops, echo, touch, Scripting

# Wann & was sollte man automatisieren?

Mehr als 4 Dokumente ...

... Umformatieren

Wiederkehrend ...

... Zusammenführen

... Extrahieren

... Statistiken

... Backup & Archivierung

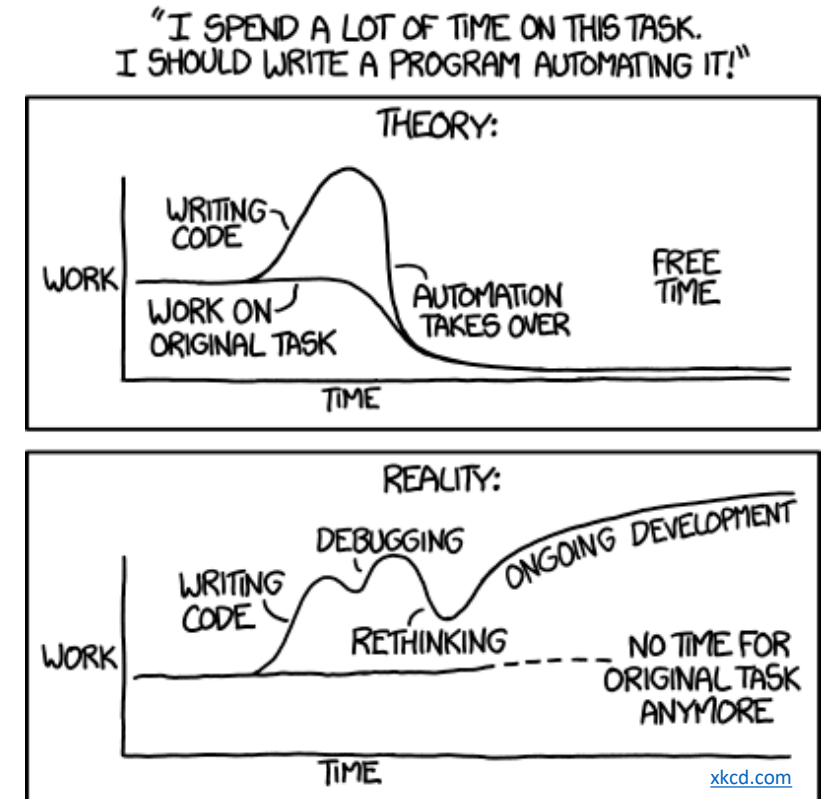
... Down-/Uploaden

Aufwand-Nutzen-Abwägung ...

... je häufiger um so einfacher!

# Was hab ich davon?

- Workflow Dokumentation
- Weniger (Schussel)Fehler
  - einen vergessen, unvollständig, ...
- Gemeinsame Entwicklung/Nutzung/Pflege
- Ggf. mehr Zeit (wenns mal läuft)





# Wie gehe ich vor?

**Schritt 1** = Problem nochmal durchdenken und **Wiederholung finden!**

Endziel:

*„ ... will ein CSV mit den DOIs **aus jeder .bib Datei** ...“*

Umformuliert:

*“... **extrahiere aus jeder .bib Datei** die DOIs  
und fasse diese im CSV Format zusammen ...”*

# Wie gehe ich vor?

`$ grep` = RegEx-basierte Zeilen-/Textextraktion

`$ tr` = Buchstabenersetzung/-löschung

Schritt 2 = **Workflow** „für EINMAL“ entwickeln!

(1) Notwendige Zeilen finden

```
/c/Lehre/DataBASHing/bash-workshop/getDoiForBib.all (bash-workshop)  
$ grep -P "(year|doi)\s*=[^,]+," Raden-2019.bib  
year={2020},  
doi={10.1186/s12859-019-3143-4},
```

(2) Daten extrahieren

```
/c/Lehre/DataBASHing/bash-workshop/getDoiForBib.all (bash-workshop)  
$ grep -P "(year|doi)\s*=[^,]+," Raden-2019.bib | grep -oP "(?<={).+(?=})"  
2020  
10.1186/s12859-019-3143-4
```

(3) Zeilenumbrüche ersetzen

```
/c/Lehre/DataBASHing/bash-workshop/getDoiForBib.all (bash-workshop)  
$ grep -P "(year|doi)\s*=[^,]+," Raden-2019.bib | grep -oP "(?<={).+(?=})" | tr "\n" ";"  
2020;10.1186/s12859-019-3143-4;
```

Einzelschritte mit „|“ in eine „pipe“ verbinden = Ausgabe wird Eingabe des nächsten Programms

# Wie gehe ich vor?

## Schritt 3 = Skriptdatei anfangen und Platzhalter einführen

SHEBANG = Welches Programm aufrufen?!

Variablendefinition

```
1  #!/usr/bin/bash
2
3  # aktuelle Datei
4  BIBFILE="Raden-2019.bib"
5
6  # Arbeitsschritte für eine Datei
7  grep -P "(year|doi)\s*=[^, ]+," $BIBFILE | \
8  grep -oP "(?<={) .+ (?=})" | \
9  tr "\n" ";"
10
```

Variablenverwendung

Zeilenumbruch  
ignorieren

# Wie gehe ich vor?

Schritt 4 = Verallgemeinerung auf beliebige Dateien

Schleife

```
1  #!/usr/bin/bash
2
3  # alle .bib Dateien im aktuellen Verzeichnis
4  for BIBFILE in *.bib;
5  do
6      # Arbeitsschritte für eine Datei
7      grep -P "(year|doi)\s*=[^,]+," $BIBFILE | \
8          grep -oP "(?<={}) .+ (?=})" | \
9          tr "\n" ";" # alle Zeilenumbrüche ersetzen
10     echo # neuer Zeilenumbruch
11 done > year-doi.csv  Ausgabeumleitung in Datei
```

# Wie gehe ich vor?

## Schritt 5 = Anwenden und testen

Skriptaufruf

```
qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBASHing/bash-workshop/getDoiFor
$ ls
Hadjeras-2023.bib Raden-2019.bib getDoiForBib.all.sh*
```

```
qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBASHing/bash-workshop/getDoiFor
$ bash getDoiForBib.all.sh
```

```
qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBASHing/bash-workshop/getDoiFor
$ ls
Hadjeras-2023.bib Raden-2019.bib getDoiForBib.all.sh* year-doi.csv
```

neue  
Ausgabedatei

Inhaltscheck

```
qubsq01@DEZ-Martin MINGW64 /c/Lehre/DataBASHing/bash-workshop/getDoiFor
$ cat year-doi.csv
2023;10.1093/femsm1/uqad012;
2020;10.1186/s12859-019-3143-4;
```

Ergebnis = ☒ wiederholbar ☒ nachvollziehbar ☒ generalisierbar

# Zusammenfassung

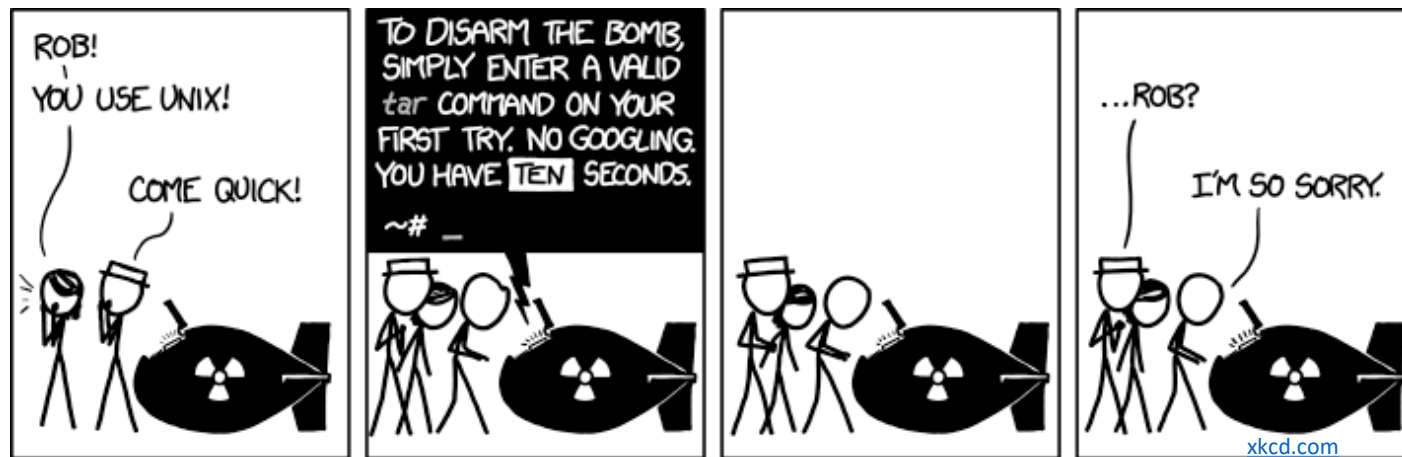
# Was nehmen sie (hoffentlich) mit ...

... die Shell kann Arbeitsprozesse vereinfachen und beschleunigen

... ist eigentlich gar nicht so kompliziert

... werde ICH mit einem [Online-Tutorial](#) nochmal vertiefen! 😊👍

... und bei Fragen wende ich mich an die Herren [Wust und Raden](#)!





# Weitere Anwendungsbeispiele ...

<https://github.com/Dr-Eberle-Zentrum/DataBASHing/blob/main/workshop/use-cases.md>

[Image by Damian Zaleski](#)

