Before we start

- create a github account (or login)
- create a project: **docker**
- start a terminal with **bash**

Bashing

Eine kleine Einfuehrung in die Bash

Author: stefan@koospal.de

Java Land 2019

Mission

Der Workshop versucht dem Anfänger der Bash-Programmierung ein paar Klippen und deren Umschiffung aufzuzeigen. Alles was man leicht bei Google finden kann, weil man von selbst die richtige Frage stellt, wird nur kurz gestreift. Schwerpunktmäßig werden Fragen beantwortet, die man selbst nicht stellen würde, weil man nicht weiß, dass man die Antwort braucht.

Also

- Keine ewigen Wahrheiten
- Keine abgefahrenen Tricks
- Für Profis wenig Neues
- Nur ein paar Tips zum Überleben

Für wen schreibe ich?

- Für mich oder andere?
- Interaktiv oder batch?

- Erste Annahme:für mich
- Mehr später ...

Anfang

- Man nehme einen Editor

TYPE
#!/bin/bash

- In die erste Zeile

Wirklich ?

Wir haben einen Vorgang

- Tippen immer wieder die gleichen Befehle ins Terminal
- Geht das nicht besser?

.bash_history

- Guter Anfang!
- Aber nicht von Dauer!
- Also skript:
  Type:
  bash

Beispiel: git

- **mkdir $HOME/bin**
- **mkdir $HOME/git**
- **cd $HOME/git**
- **git clone https://github.com/koospal/docker**
- **cd docker**
- **touch Dockerfile**
- **git add Dockerfile**
- **git commit**
- **git push**

.bash_history - skript

Type:

```
bash exit
tail $HOME/.bash_history
  > $HOME/bin/chggitproject
chmod +x
    $HOME/bin/chggitproject
```

Umgebungsvariablen

- **echo $HOME**
- **echo $EDITOR**
- **$EDITOR $HOME/bin/chggitproject**

Skeleton - Minimalform

```
#!/bin/bash

#main
```

Wozu das Ganze?

Vorgänge auf der Commandline automatisieren!

Wie?

Meist **Quick and Dirty**!

1) Auf der Commandline ausprobieren
2) In das Skeleton kopieren
3) Verfeinern

Skeleton mit Parameter

```
#!/bin/bash

#parameter check

#main
```

Parameter

```
${1} .. ${10} ..
$0 = Skriptname wie aufgerufen
$# = Anzahl der Parameter

$$ = Prozessid
```

Variable

bestehen aus:
Bezeichner
  = Zeiger auf Speicherstelle
Wert
  = NULL terminated String
  Auswertung durch $

Zuweisung durch =

Variable auslesen

```
BINDSP=”dsp”
echo ${BINDSP}
echo ${BINDSP}log
```

Normalerweise alle global

Skeleton mit USAGE

```
#!/bin/bash
USAGE="$0 parameter1"
NUMBPARA=1
#Kein Leerzeichen vor und nach =
#parameter check
if [ $# != $NUMBPARA ]; then
    echo $USAGE
    exit 1
fi
#main
```

Skeleton Umgang Parameter

```
#assign parameter
project=$1
#main
echo kom: using ${project}
```

Skript main

```
mkdir $HOME/git > /dev/NULL 2>&1
cd $HOME/git
git clone \
https://github.com/koospal/${project}
cd docker
$EDITOR Dockerfile
git add Dockerfile
git commit
git push
exit 0
```

stdin,stdout,stderr,pipe

wie in C
- stdin <
- stdout >
- stderr 2

Pipes |

stdin,stdout,stderr,pipe

Type:

```
ls  > /tmp/files.lst
cat < /tmp/files.lst
mkdir /tmp/abc
mkdir /tmp/abc >/tmp/o 2>&1
ls -l |wc -l
```

Achtung: Betriebssystem !

Variablen zuweisen

Type:

`NO_OF_FILES=`ls -l |wc -l``

- Standardout einer Befehlsfolge

Type:

`read PRAEFIX`

- read vom Stdin

for mit Liste

Type:

```
echo *
for i in *; do
    echo $i
done
```

*, ? und Listen

- * und ? fuer Dateilisten

```
echo a?b*c
```

- Listen aus Textdateien

```
cat /tmp/files.lst
```

- erzeugte Listen

```
`echo *`
```

for zaehlschleife

- Type:

```
for ((i=1;i<10;i++));do
    echo $i
done
```

Bsp: Logfiles auswerten

- Es gibt viele Tools
  z.B. Webalizer
- Aber manchmal passen die nicht

Logfiles Beispiel

https://java.de/b/log.csv
Oder git
Aufgabe:
 cups logs auswerten

Commandlineprogramme

- **ls, echo, mkdir, cd**
- **git**
- Werden mit Parametern aufgerufen
- Input oft von stdin
- Output meist auf stdout

cat

- Ein-/Ausgabe einer Datei
- Oft ein guter Anfang

cat log.csv

grep

- Ausgabe von Zeilen einer Textdatei, in der ein Muster vorkommt

```
cat log.csv|grep total
```

Nur einige Spalten ...

```
20 +0100] total 2
27 +0100] total 0
Nur != 0
```

```
...|awk '{if ($10 != 0)
```

awk

Achtung noch eine Programmiersprache!

- Zeilenweise Bearbeitung von strukturierten Textstreams
- Das “Schweizer Messer”
- Solaris/MAC/usw: evtl. **gawk**

```
print $6,$4,$5,$10}'
```

sort

Zeilenweise sortieren
von Textstreams

|sort -n -u

awk mit BEGIN und END

- Initialsierung und Abschluss

```
|awk '\
      BEGIN {z=0}\
      {print $0;z=z+$4}\
      END{print "sum",z}\
  '
```

Einbau ins Skeleton

Entscheidung:

- Mit Parameterbehandlung
- **cat $1|**

oder

- nur als Filter

Weshalb **cat** ?

- Erleichtert den Einbau weiterer Filter

Bsp: Daten von Webseiten

Vorteile **bash** :

- Schnelle Lösung
- Nicht 100%

Alternativen bedenken:

- Andere Sprache?
- Fertige Tools?

Emailadressen sammeln

https://www.uni-math.gwdg.de/staff/v2/mitarbeiter.html

Oder

https://java.de/b/web.html

curl -k
https://www.uni-math.gwdg.de/staff/v2/mitarbeiter.html

Oder auf github

cat und **grep**

cat web.html|grep @

- Mal sehen, was geht

```
<td><a class="hplink"
href="http://www.uni-math.gwdg.de/tammo/" target="_top">tom
Dieck, Tammo</a>, Prof. Dr.<br /><span class="email">
tammo.tom-dieck@mathematik.uni-goettingen.de</span></td>
```

|grep "class=\"email"

sed

- Streameditor – Verarbeitung von Textstreams
- z.B. um Störendes zu entfernen

```
|sed -e "s/<td>.*email//g"
```

- Nutzt Regular Expressions

awk mit Fieldseparator

- Zum einfachen extrahieren

```
|sed -e "s/<\/span//g"\
|awk -F">" '{print $2}'
```

- **F** kann regualar Expression sein

Batchbetrieb

- Achtung Falle: **PATH** !
- Programme, die beim USER im Pfad sind, sind es nicht unbedingt bei root
- Je nach Pfad werden verschiedene Versionen der Programme genutzt: solaris-awk oder gnu-awk

Bsp.: Konvertierung

- ics → csv → txt-Liste
- http://java.de/roller/blog/page/user_group_treffen
  →
- http://java.de/ijugtermine.txt

Ein Vorschlag für Kommentare

```
COMM="echo comment: "
COMM=":"
NOTE="echo note: "
DEBUG="echo debug: "
DEBUG=":"
```

: bedeutet eingentlich true

Etwas Pseudocode

```
urlf=icsurl.csv

$COMM create url list
$COMM loop all urls
for url in `awk -F";" '{print $2}' $urlf`; do
    $NOTE get ics file $url
    $COMM convert ics file to csv file
done
$COMM sort all entries by date
$COMM loop all entries
```

Fehlerbehandlung

```
for url in `awk -F";" '{print $2}' $urlf`; do
    $NOTE get ics file $url
```

- **Was tun bei einem Fehler?**

```
done
```

Vorschlag: Funktionen

```
ERRORSUM=0
ERRORVALUES=""
ERRORTEXTS=""

errorhandler () {
local lasterror=$1
local behave=$2
local errortext=$3
$COMM "$1=errorvalue
$2=INIT/CONT/ENDE/STOP $3=text"
}
```

Lokale Variable

- Innerhalb eines Blocks {}
- Vor allem in Funktionen

Unbound Variables

echo ${NOINITIAL}
oder Schreibfehler
echo ${NO_OF_FILE}

set -u

- **Skriptabbruch bei unbound Variable**

date – Datum und Zeit

```
local TIMESTAMP=`date +"%Y%m%d%H%M%S"`
```

- Kann viel
- Achtung: Systemabhängig

Linux: **date --date=100 "+%s"**

MAC: **date --date=100 "+%s"**

case – der Schalter

```
case "$behave" in
    "INIT") $NOTE $TIMESTAMP $errortext;return $lasterror;;
    "CONT") return $lasterror;;
    "ENDE") if [[ $lasterror == 0 ]];then return
$lasterror;fi ;;
    "STOP") $NOTE $TIMESTAMP $errortext;;
    *) ;;
esac
```

- Kann auch regular Expressions

if - Bedingung

```
if [ $lasterror == 0 ];then return $lasterror;fi
```

- Eigentlich steht da:

if test $lasterror == 0 bzw. **if test $lasterror -eq 0**

- Hauptfehler:Leerzeichen

```
if [[ $lasterror == 0 ]];then return $lasterror;fi
```

- Tipp[[ ]],== usw. Nutzen
- Man kann sehr viel testen
  z.B. Ist es eine leer Datei?

Rechnen

```
ERRORSUM=$((ERRORSUM+lasterror))
```

Ist doch ganz einfach:

```
let a=1+2
a=$((1+4))
```

Integerarithmetik

sizeof(int),also 64bit

Rechnen mit float

```
LANG=de_DE.UTF-8;printf "%f" 2
LANG=en_US.UTF-8;printf "%f" 2

LANG=en_US.UTF-8
echo 1.1 2|\
awk '{printf "%2.2f",$1*$2}'
```

Es gibt auch **bc**

Weshalb so?

- Ich muss mir nur eine Syntax merken (awk)
- Formatierung eingebaut
- Variable, die nicht als Zahl interpretiert werden kann, wird 0

Weshalb will ich das?

- Variable, die nicht als Zahl interpretiert werden kann, wird 0

```
a=1
if [[ a != 0 ]]; then echo ja;fi
if [[ $a != 0 ]]; then echo ja;fi
a=1.5
if [[ $a > 0 ]]; then echo ja;fi
if [[ $a > 1.1 ]]; then echo ja;fi
```

exit Fehlerstatus

Sollte das letzte Kommando jedes Skriptes sein.

“Libaries”

- Die errorfunctions und die initfunctions packe ich in ein “Libary”

```
#!/bin/bash

. /opt/bin/initfunctions
. /opt/bin/errorfunctions
COMM=":"
errorhandler 0 INIT $0
```

Große Zahlen

- Pythonskript aufrufen
- Beispiel IBAN

Arrays

```
declare -A dayoftheweek
dayoftheweek["Mon"]="mo"
dayoftheweek["Tue"]="di"
dayoftheweek["Wed"]="mi"
dayoftheweek["Thu"]="do"
dayoftheweek["Fri"]="fr"
dayoftheweek["Sat"]="sa"
dayoftheweek["Sun"]="so"
```

- Assoziativer Arrays

Warten

sleep <Sekunden>

wait <prozess id>

```
startbackroundcmd &
pid_startbackroundcmd=$!
```

Kopf und Fuss

```
head  <textfile>
 head -1 tabelle.csv
tail  <textfile>
 tail -f /var/log/syslog
```

Wo und was

```
which <command>
 which echo
file <list of files>
 file `which echo`
find <dir> [options]
```

Dateien jonglieren

diff <file1> <file2>
join <s-file1> <s-file2>
tar <options> <tarfile>
- Warnung vor cp -R
- besser tar

Endlos und bedingt

Endlosschleife

```
while [[ 1 ]]; do
    echo x;read
done
```

Bedingte Ausführung

```
ls log.csv && echo ja
```

Viele Parameter - **shift**

#check all files in /bin

cd bin;chkallf *

```
#!/bin/bash
set -u
if [[ $# < 1 ]]; then
 echo zu wenig Parameter
fi
while [[ $# > 0 ]]; do
 file $1
 shift
done
```

Portabel Programmieren

```
awk=/usr/bin/gnu/awk

cat log.csv| $awk ...
```

```
awk Systemcalls

awk \
'{dir="/bin";\
cmd="ls "dir;\
system(cmd)}'
```

vi – manchmal ist er da

ESC (vielleicht mehrmals)
→
Befehlsmodus
Dann:
:
q!

bash ≠ sh

bash v3 ≠ bash v4 ≠ bash v5

[[ ]] (( ))
gibt es nur in der bash

Links:

http://openbook.rheinwerk-verlag.de/shell_programmierung/

https://www.uni-math.gwdg.de/koospal/website/vortraege/

Variable und Typen

Varibalen sind Bezeichner,
die auf eine Speicherstelle
im Hauptspeicher zeigen.
Der Wert ist die Bitfolge
an der Speicherstelle.
Typ ist immer String.