



Über mich



- Tam HANNA
- Tamoggemon Holding k.s.
- Instagram: tam.hanna
- Twitter: tamhanna

Vortrag wird GETAPED!

BITTE TAPE ANSEHEN!!!

Slides sind alleinstehend verwirrend!!!

Zielgruppe

- Unix-Nutzer
 - Kommandozeilentools sind Basis
- GRUNDLEGENDE Programmierkenntnisse
 - C, Python, Java, C# etc. sind ideal
 - Alles andere “geht auch”

Inhaltsübersicht

- Was ist die Shell?
- Was ist ein Shellskript?
- Fortgeschrittenes Scripting
- Shellskripte, grafisch
- Fehlersuche



Zu Ehren des Logins

- Shell == Programm, das login anwirft
 - Oder anderswie startet
- Verarbeitet Benutzereingaben
 - Interne Kommandos (z.b. cd)
 - Externe Kommandos (aus PATH)

```
tamhan@tamhan-thinkpad: ~/Downloads/android-studio/bin
Support: command not found
tamhan@tamhan-thinkpad:~$ cd Downloads/
tamhan@tamhan-thinkpad:~/Downloads$ cd android-studio/
tamhan@tamhan-thinkpad:~/Downloads/android-studio$ ls
bin          gradle          jre  license      NOTICE.txt
build.txt    Install-Linux-tar.txt  lib  LICENSE.txt  plugins
tamhan@tamhan-thinkpad:~/Downloads/android-studio$ cd bin
tamhan@tamhan-thinkpad:~/Downloads/android-studio/bin$ ls
appletviewer.policy  idea.properties  printenv.py      studio.sh
format.sh            inspect.sh       restart.py        studio.voptions
fsnotifier           lldb            studio64.voptions
fsnotifier64         log.xml         studio.png
tamhan@tamhan-thinkpad:~/Downloads/android-studio/bin$ ./studio.sh
```


Mehr Shells

- Verschiedene Shells existieren
 - csh
 - bash
 - dash
 - sh
- **Teilweise inkompatibel zueinander**

Shell in Shell

- Shell ist ein **PROGRAMM**
- Shell-*Instanz* ist ein **PROZESS**
- Eine Shell kann andere Shells anwerfen
- Shell-Eingaben können nicht nur direkt sein
 - TCP/IP
 - Pipes, etc...

Wieso Scripting

- Systemtechnische Definition
 - Shell-Skripte werden nicht kompiliert
- Meine Definition
 - Wer in BASH Türme von Hanoi codet, ist irre...
 - Weil: Masochist

Ein motivierendes Beispiel

- Aufsetzen komplexer Server

```
provisioner.sh x
sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common
sudo apt-get update
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
EOF

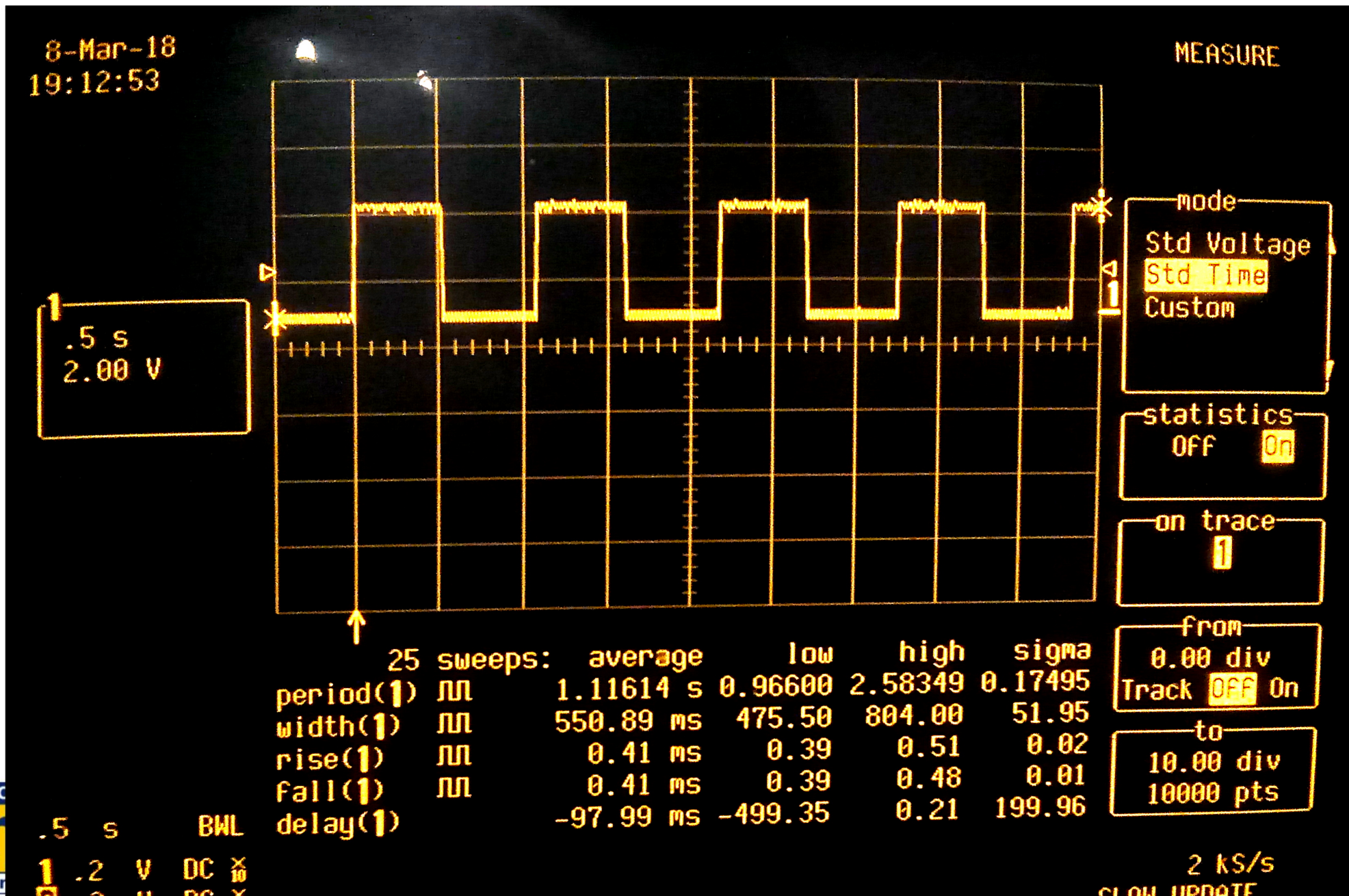
sudo apt-get update
sudo apt-get install -y docker.io
sudo apt-get install -y kubelet kubeadm kubectl kubernetes-cni
shutdown -r now
```

Und noch eins...

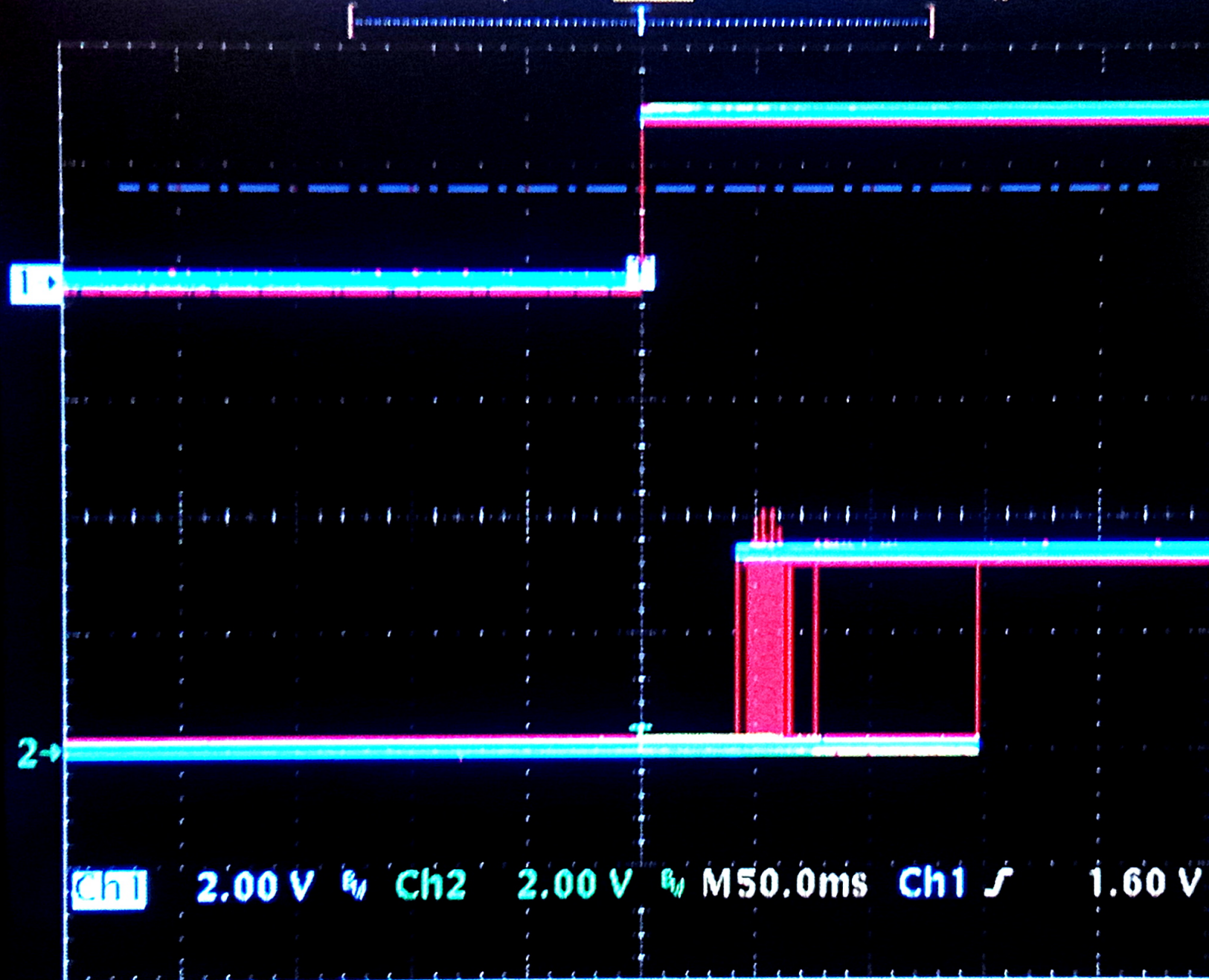
- Prüfung des Verhaltens eines IoT-Gateways
- Permanentes Absetzen von Befehlen via CURL

```
#!/usr/bin/env bash
while :
do
    curl 'https://tamoggemon.mozilla-iot.org/things/gpio-18/p'
    curl 'https://tamoggemon.mozilla-iot.org/things/gpio-18/p'
done
```

...samt Messergebnissen



Tek Run: 1.00MS/s Sample DPO DPO Brightness: 50 %



Cursor
Function

Off

H Bars

V Bars

Paired

Function
Off

Mode
Indep

Time
Units
1/sec (Hz)

Amplitude
Units
Base



Vorbereitendes

- Shellsript: *.sh
- Erstellen einfach:

```
tamhan@tamhan-thinkpad:~/Desktop/Linuxtage/ShellProgBeispiele$ clear  
tamhan@tamhan-thinkpad:~/Desktop/Linuxtage/ShellProgBeispiele$ touch 0.sh  
tamhan@tamhan-thinkpad:~/Desktop/Linuxtage/ShellProgBeispiele$ gedit 0.sh
```

- Ausführen: nur mit EX-bit

```
tamhan@tamhan-thinkpad:~/Desktop/Linuxtage/ShellProgBeispiele$ ./0.sh  
bash: ./0.sh: Permission denied  
tamhan@tamhan-thinkpad:~/Desktop/Linuxtage/ShellProgBeispiele$ chmod +x 0.sh  
tamhan@tamhan-thinkpad:~/Desktop/Linuxtage/ShellProgBeispiele$ ./0.sh
```

Umleitungen

- Weil es vorher gefordert wurde

```
tamhan@tamhan-thinkpad:~/Desktop/Linuxtage$ dmesg
```

```
...
```

```
[13743.892830] wlan0: authenticated
```

```
[13743.893041] wlan0: waiting for beacon from 38:0e:4d:bf:0e:ca
```

```
[13743.997385] wlan0: associate with 38:0e:4d:bf:0e:ca (try 1/3)
```

```
[13743.999711] wlan0: RX AssocResp from 38:0e:4d:bf:0e:ca (capab=0x111 status=0 aid=29)
```

```
[13744.018446] wlan0: associated
```

```
[13761.146219] usb 1-1.4: USB disconnect, device number 3
```

```
[14281.126064] systemd-hostnamed[26011]: Warning: nss-myhostname is not installed. Changing  
the local hostname might make it unresolveable. Please install nss-myhostname!
```


Umleitungen - II

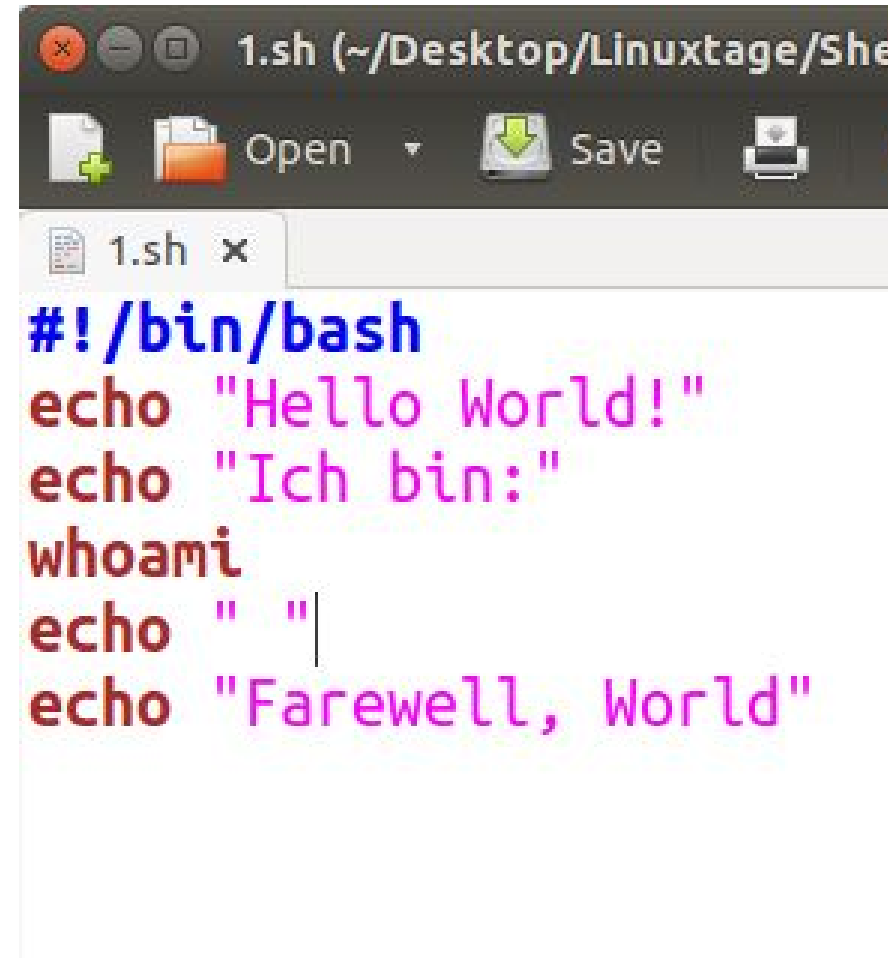
- Grep
 - Matcht gegen “regular expressions”
 - Eliminiert nicht passende Eingabeteile
 - Ideal als Verwerter für dmesg
-
- LÖSUNG: PIPE

Umleitungen - III

```
tamhan@tamhan-thinkpad:~/Desktop/Linuxtage$ dmesg | grep "ttyU"  
[10884.581442] usb 1-1.2: FTDI USB Serial Device converter now attached to tttyUSB0  
[12390.871685] ftdi_sio tttyUSB0: FTDI USB Serial Device converter now disconnected from ttty  
tamhan@tamhan-thinkpad:~/Desktop/Linuxtage$
```

Ein Beispiel

- Shebang
- Befehlssequenz



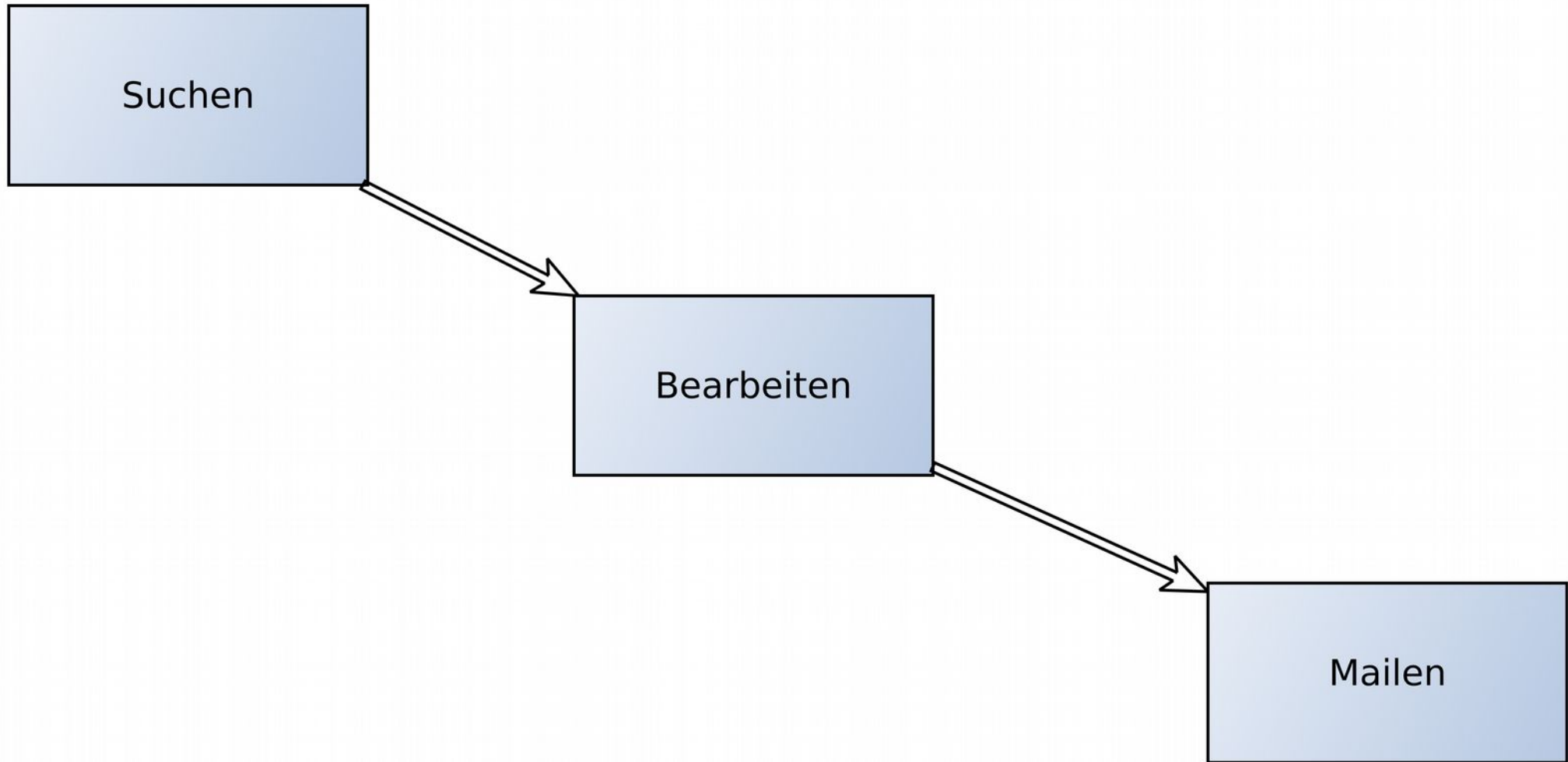
The screenshot shows a terminal window titled "1.sh (~/Desktop/LinuxTage/She...". The window has a menu bar with "Open", "Save", and a printer icon. The script content is as follows:

```
#!/bin/bash
echo "Hello World!"
echo "Ich bin:"
whoami
echo " "
echo "Farewell, World"
```

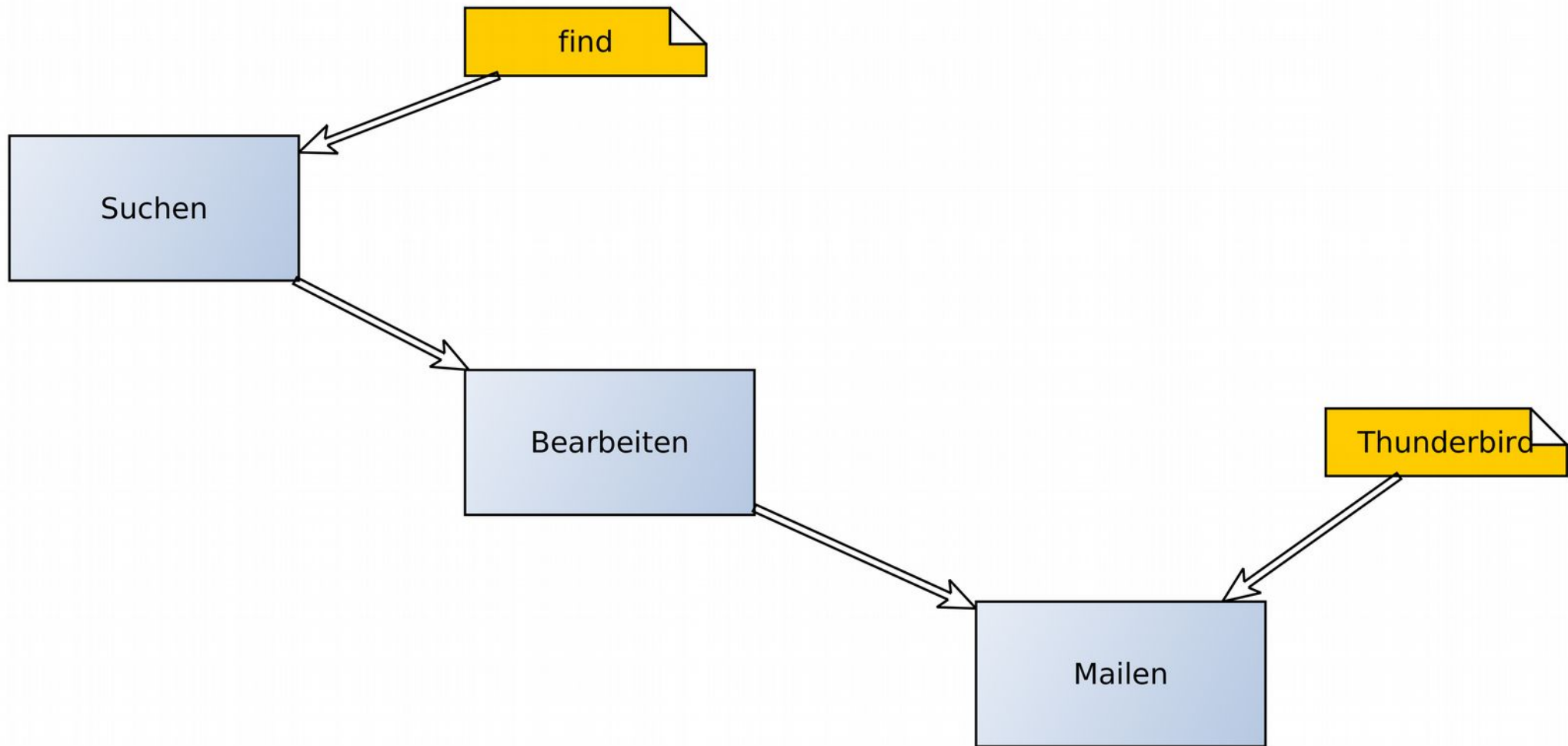

Wo Bash brilliert...

- Anwalt tippt SEHR langsam
- Gedanke: sprich auf Band!
- Sende per e-mail an Transkribeur
- Wahre Aufgabe!!!

Aufgabe in Einzelteile zerlegen



Systemkenntnisse wichtig!



Daten umleiten

- Linux-Programm ist “Kanalverarbeiter”
 - STDIN
 - STDOUT
 - (STDERR)
- Bestes Beispiel: Pipes

Variable

- Speicherfeld für beliebige Daten
- Schreiben “wie gewohnt”
- Nutzung des Werts per \$

```
2.sh x
#!/bin/bash

willkommen="Hallo, Schlomo!"
echo " - - - - -"
echo "\$willkommen"
echo $willkommen
echo " - - - - -"
echo "willkommen"
echo willkommen
echo " - - - - -"
echo " ACHTUNG, FALLE!!"
echo "$willkommen"
echo $willkommen
echo " - - - - -"
```

Achtung, Falle!

- Das Dollarzeichen ist immer erforderlich
- tester würde hier “willkommen” enthalten

```
tester=willkommen  
echo $tester
```

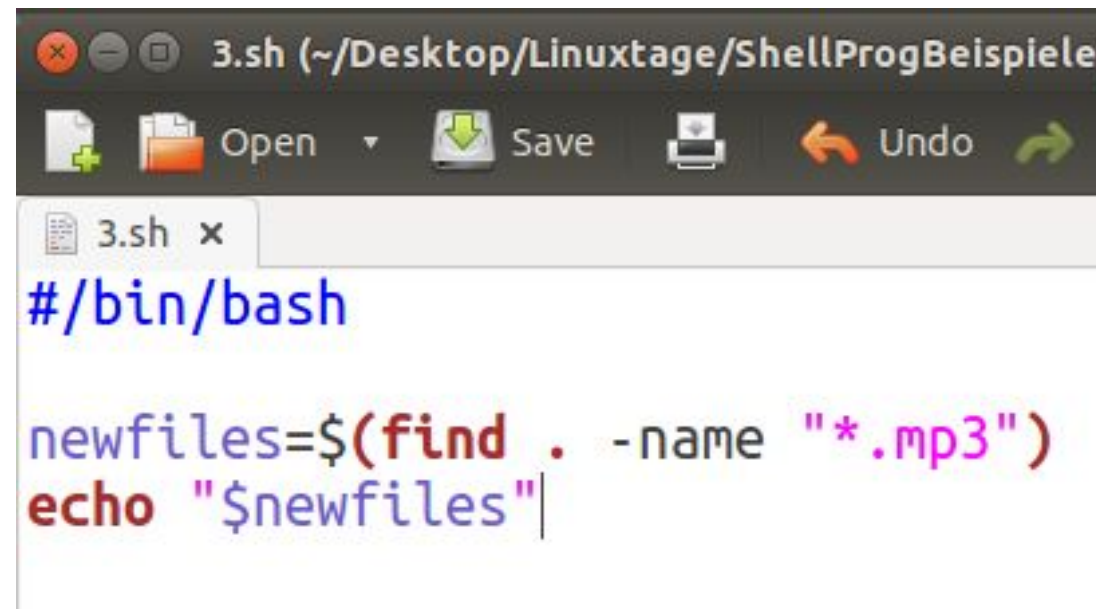
-
- BASH wirft KEINEN (!!!) Fehler

Ein Array...

- Find, etc sind schizophren
- Verschiedenes Verhalten nach STDOUT
 - Terminal
 - Pipe

Rein und Raus....

- `$()` erlaubt “Einsammeln” von Daten
- Ergebnisse von Find wandern in Variable
- Parametrierung: via Kommandozeile

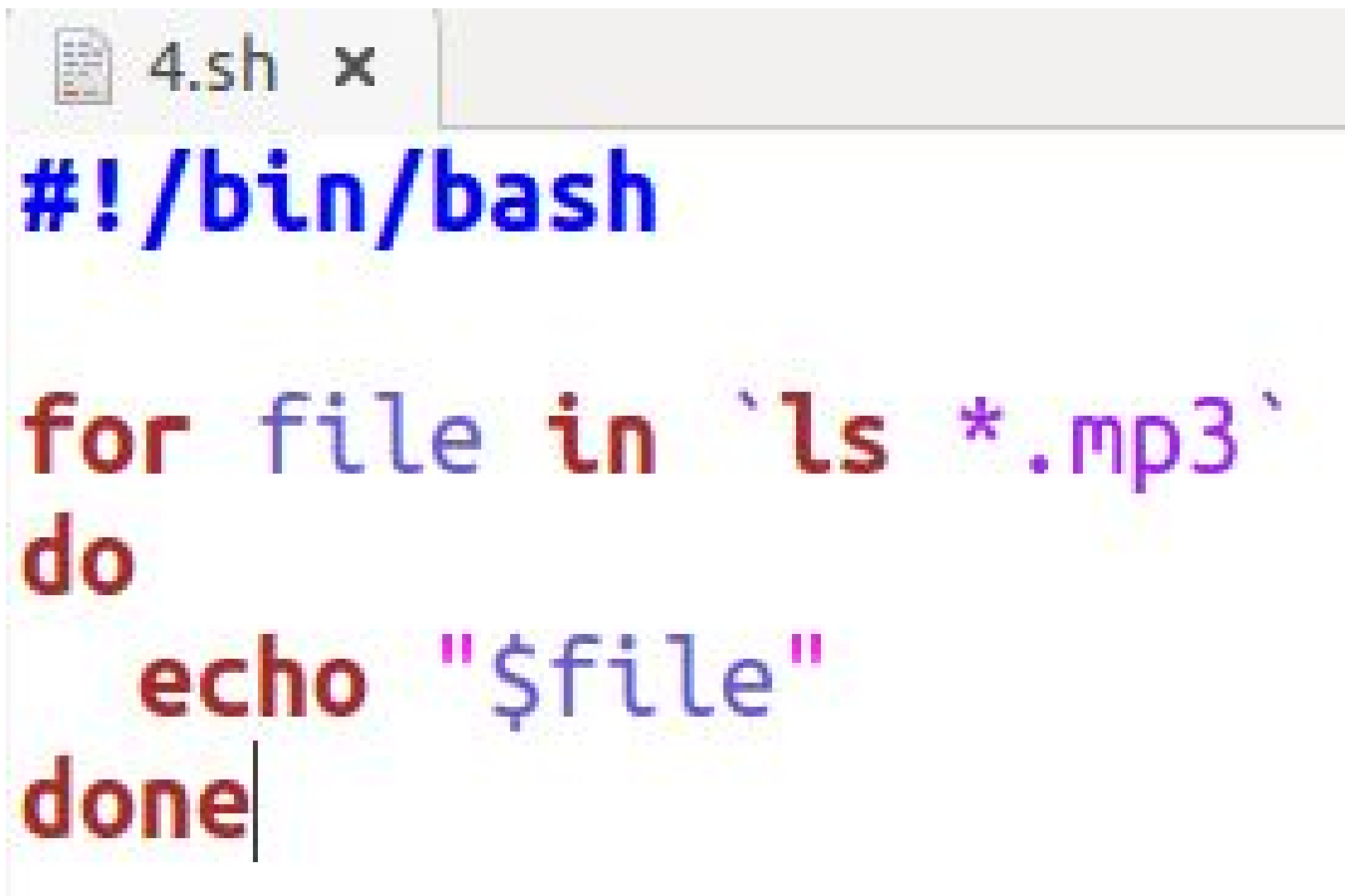


The screenshot shows a terminal window titled "3.sh (~/Desktop/LinuxTage/ShellProgBeispiele)". The window has a menu bar with "Open", "Save", and "Undo" options. The terminal content shows a shell prompt `#!/bin/bash` followed by two lines of code: `newfiles=$(find . -name "*.mp3")` and `echo "$newfiles"`. The code is color-coded: `find` is red, `.` is blue, `-name` is green, `"*.mp3"` is pink, and `echo` is red.

```
#!/bin/bash

newfiles=$(find . -name "*.mp3")
echo "$newfiles"
```


Stein um Stein

A terminal window with a title bar containing a document icon, the text '4.sh', and a close button 'x'. The terminal content shows a bash script with syntax highlighting: the shebang is blue, the loop keywords are red, the file list command is purple, and the echo command is red. A vertical cursor is at the end of the 'done' line.

```
#!/bin/bash

for file in `ls *.mp3`
do
    echo "$file"
done
```

Back-Ticks

- “Evaluere dies”
- Ersetze den Inhalt durch die Ausgabe
- Arbeiten auf textueller Ebene

Er spinnt

- Wir haben nur wenige MP3-Files
- Aber er “bricht auf”
- An den Leerstellen...
- Ursache: Backticks (don't use them)
 - <https://unix.stackexchange.com/questions/27428/what-does-backquote-backtick-mean-in-commands>

Lösung: Quoting

```
5.sh x
#!/bin/bash

for file in "$(ls *.mp3)"
do
    echo "$file"
done
```

Quoting - II

- “...Alles, außer \$ schützen
echo "\$SHELL"
- '...killt alles
echo '\$SHELL'

Und jetzt mit Zähler

```
6.sh x
#!/bin/bash

fileCount=0
for file in "$(ls *.mp3)"
do
    fileCount=$((fileCount+1))
done

for file in "$(ls *.mp3)"
do
    echo "Email-Sender: File $fileNow of $fileCount"
    fileNow=$((fileNow+1))
    echo "$file"
done
```

Schleife ohne Ende

- Scheitert immer noch
- \$file wird in EINEM Durchlauf abgearbeitet
- Shell teilt Resultat wegen “” nicht auf

Trenne weniger

```
7.sh x
#!/bin/bash

IFS=$'\n'
fileCount=0
for file in `ls *.mp3`
do
    fileCount=$((fileCount+1))
done
```

Und nun raus damit!

```
*8.sh x
#!/bin/bash

echo "Enter a title, please!"
read emailTitle

fileCount=0
for file in `ls *.mp3`
do
    fileCount=$((fileCount+1))
done

fileNow=1
for file in `ls *.mp3`
do
    emailsubject="$emailTitle File $fileNow of $fileCount"

    fullpath="$(readlink -f $file)"
    thunderbird -compose "to='e@e.com',subject='$emailsubject',attachment='$fullpath'" &

    fileNow=$((fileNow+1))
done
```



Wikimedia Commons //
Lothar Willmann

Suche den Tam

```
#!/bin/bash

while true; do
    VAR=$(hcitool scan)
    if [[ $VAR == *"10:2A:B3:B2:8E:D8"* ]]
    then
        logger "Tam is likely to be around" -t "IMAGINE";
    fi
    sleep 60;
done
```

logger und mehr

- Logger schreibt Nachricht ins syslog
- [[]]
 - Bashismus
 - Platzierung von “Konditionalem”

Klassische Syntax: []

```
alteabfrage.sh x
#!/bin/bash
echo "Choose the material"
echo "a...Analogue electronics tutorial!"
echo "g...Review of gadget"
echo "r...Review of book!"

read -n 1 emailShorty
echo "  "
```

Aufbrechen des Inputs

```
if [ $emailShorty == "a" ]; then
    echo "Alert, electronics head detected"
    emailTitle="Analogue electronics tutorial!"
elif [ $emailShorty == "g" ]; then
    emailTitle="Review of gadget"
elif [ $emailShorty == "r" ]; then
    emailTitle="Review of book"
else
    echo "Error"
    exit
fi
```

Funktionen

- BASH

```
function foo() {  
    echo "foo"  
}
```

- POSIX

```
bar() {  
    echo "bar"  
}
```


Funktionsbeispiel

```
#!/bin/bash

function drawline {
    echo " - - - - - "
    echo $#
    echo $1
}

drawline a b c d|
```

Automatische Ausführung

- CRON
 - Chronologische Abarbeitung der Programme
- udev
 - Durch Hardware gesteuerte Ausführung
- Init-Files
 - Beim Starten

Interdisziplinär denken

- Shell steht nicht allein
 - C
 - Java
 - Qt
 - etc, etc, etc

Java anwerfen

- BITTE: Class-File präkompilieren
- Ausführung dann mittels java

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

```
echo "Hello from Shell!"  
java MainClass|
```

Von Java nach Shell

- Kommunikation via Exit Codes

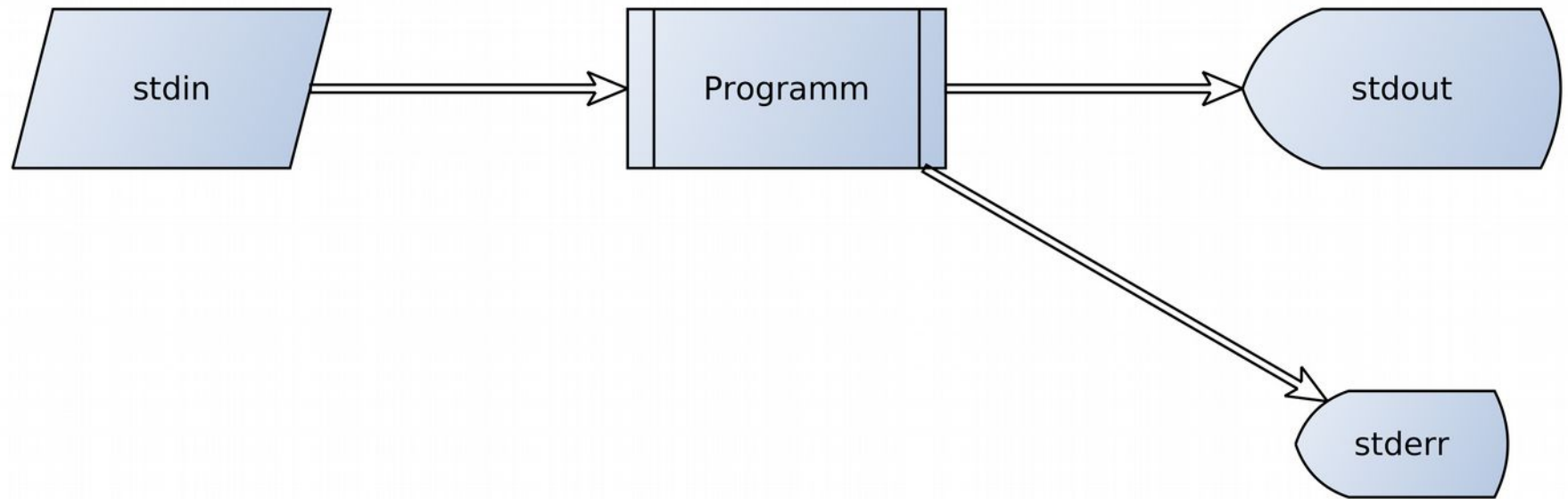
```
public class MainClass {  
  
    public static void main(String[] args) {  
        System.out.print("Enter something: ");  
        String input = System.console().readLine();  
        if(input.compareTo("something")==0){  
            System.exit(0);  
        }  
        System.exit(1);  
    }  
}
```

Auswertung des Ergebnisses

```
#!/bin/bash
java MainClass

response=$?
case $response in
    0) echo "Input OK!";;
    1) echo "Bad input";;
esac
```


Datenströme im Blick



Lesen aus STDIN

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

```
echo "Input one required"
```

```
read value1
```

```
echo "Input two required"
```

```
read value2
```

```
echo "You entered $value1 $value2"
```

Read ist nicht gierig

- \n unterbricht den ersten Read-Befehl

```
public static void main(String[] args) {  
    try {  
        Process proc = Runtime.getRuntime().  
            exec("/home/tamhan/workspace/Imagine1/bin/runner.sh");  
        BufferedWriter myWriter = new BufferedWriter(  
            new OutputStreamWriter(proc.getOutputStream()));  
        Thread.sleep(250);  
        myWriter.write("String A\nString B\n");  
        myWriter.flush();  
        proc.waitFor();  
    }  
}
```

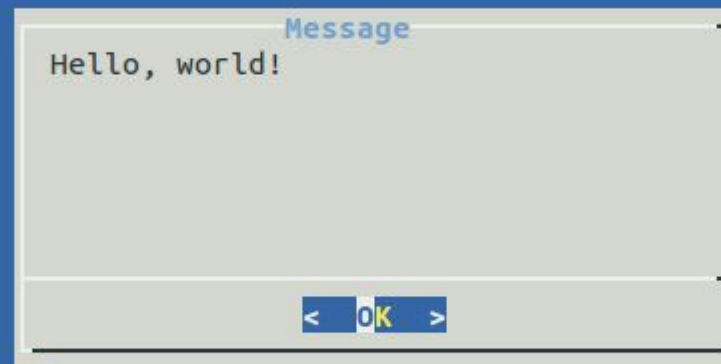



Der Mensch im Mittelpunkt

- User haben Angst vor Kommandozeilen
- Endanwender “weinen”
- Lösung: Semigrafik

Erste Meldung

```
dialog --title 'Message' --msgbox 'Hello, world!'
10 40
```



Ja oder Nein?

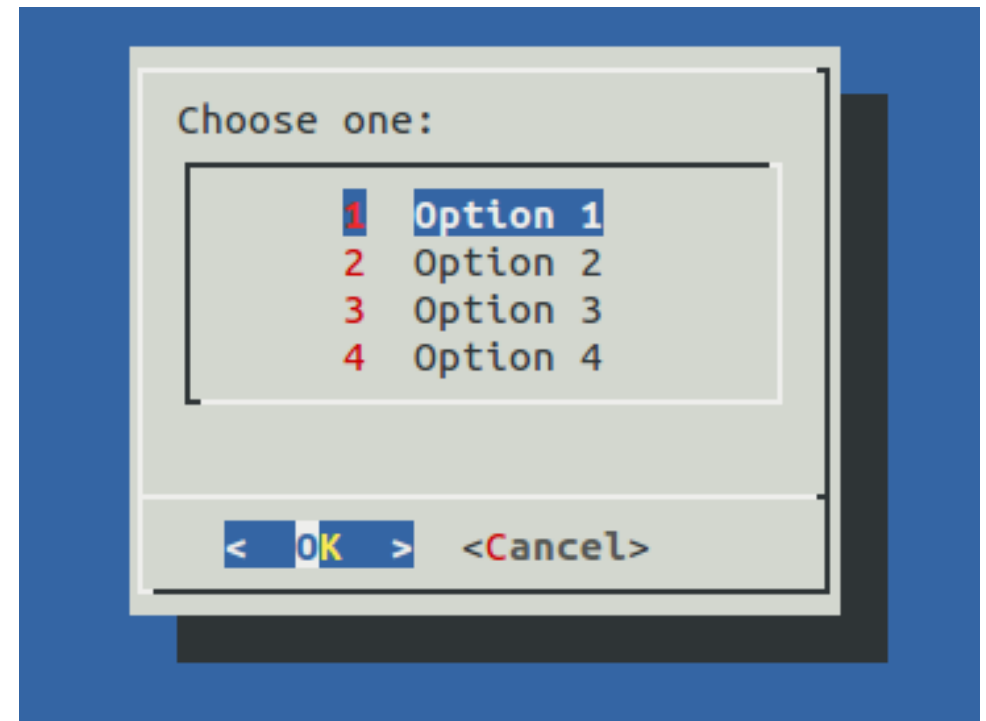
```
rocketlaunch.sh x
#!/bin/bash

dialog --title "Message" --yesno "Should we launch a missile?" 6 40

response=$?
case $response in
    0) echo "Rocket launch!";;
    1) echo "Canceled!";;
    255) echo "[ESC].";;
esac
```

Ein Menü!

- Dritter Parameter
 - Anzahl der sichtbaren Optionen
- Menüparameter sind zweiwertig
 - String
 - “Shortcut”



Ein Menü, redux

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

```
dialog --menu "Choose one:" 12 30 4 1 "Option 1" 2 "Option 2" 3 "Option 3" 4 "Option 4"  
response=$?
```

Beliebige Eingaben einsammeln

- Textuelle Eingabe, aber grafisch
- Problem: wie komme ich an die Daten ran?
- Lösung: Streams
 - 0...In
 - 1...Out
 - 2...Err
 - 3...9...Freie Spitze

Umleitung - I

- >
 - Umleitung von Ausgabe
- <
 - Umleitung von Eingabe

Umleitung - II

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

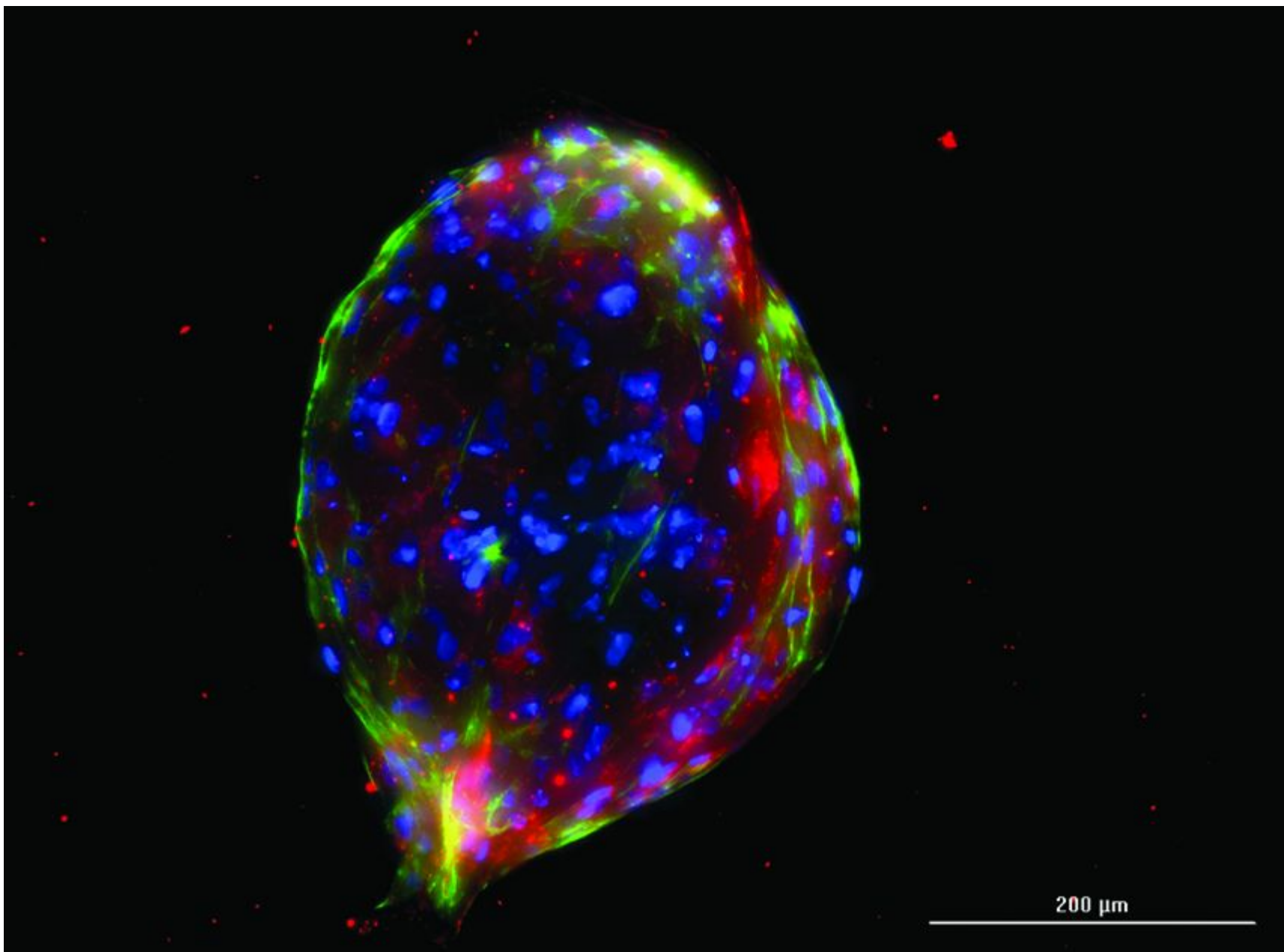
```
cache=$?
```

```
dialog --inputbox "What is your input" 10 40 2>$cache
```

```
name=$(<$cache)
```

```
echo "User input was:"
```

```
echo $name
```

Wikimedia Commons//
BioTek Instruments, Inc.

Wieso?

- Bash-Syntax ist “weich”
- Fehler tauchen erst beim Ausführen aus
- Fehler sind schwer lokalisierbar (!!!)

Grafisches Debugging

- Eclipse, **NUR** mit Oracle-JDK
 - <https://marketplace.eclipse.org/content/shell-script-dltk>
 - Window -> Preferences → Shell Script → Interpreters
 - <https://sourceforge.net/projects/basheclipse/?source=directory>
 - JAR-Files nach /home/<home>/java-oxygen/eclipse/dropins

Grafisches Debugging - II

- “Sklavenskript einbinden”
- Debuggen

```
. _DEBUG.sh  
echo $SHELL  
a=1234567890  
echo ${a}
```

- Kommunikation mittels TCP/IP (!!!)

Grafisches Debugging - III

- Visual Studio Code
 - Erweiterung aus dem Marketplace laden
 - Funktionsumfang gilt als geringer...

bashdb

- “gdb for Bash”
- Weit verbreitet
- Deployment einfach
 - `sudo apt-get install bashdb`

bashdb - II

- Syntax stark an gdb angelehnt
- step / next
- Kennt auch Breakpoints / Watchpoints

Thank you!

?!?! → tamhan@tamoggemon.com



Instagram

Search

Get the app

Sign up | Log in



tam.hanna

Follow

368 posts

7,098 followers

4,281 following

Crazy Electronics Lab Random snapshots from my #electronics lab. I love to help - questions to tamhan@tamoggemon.com please, as I sometimes miss comments!
shop.heise.de/katalog/ix-tutorial-cross-plattform-entwicklung-mit-qt

