

Bash Coding

Security Meetup

Mittwoch, 11. Mai 2016

Matthias Altmann

Überblick

1. Motivation
2. Einführung
3. Häufige Probleme
4. Best Practices / Bad Practices
5. Ausgewählte Beispiele

Motivation

- Man Pages

MANUAL SECTIONS

The standard sections of the manual include:

- 1 User Commands
- 2 System Calls
- 3 C Library Functions
- 4 Devices and Special Files
- 5 File Formats and Conventions
- 6 Games et. Al.
- 7 Miscellanea
- 8 System Administration tools and Deamons

Distributions customize the manual section to their specifics, which often include additional sections.

Motivation

▪ Öffentliche Server im Internet

Source	Date	Unix, Unix-like				Microsoft Windows	References
		All	Linux	FreeBSD	Unknown		
W3Techs	Feb 2015	67.8%	35.9%	0.95%	30.9%	32.3%	[97][98]
Security Space	Feb 2014	<79.3%	N/A				[99][100]
W3Cook	May 2015	98.3%	96.6%	1.7%	0%	1.7%	[101]

https://en.wikipedia.org/wiki/Usage_share_of_operating_systems#Public_servers_on_the_Internet, abgerufen 26.4.2016

97. ^ "Usage of operating systems for websites". W3Techs. 7 March 2015.
98. ^ "Usage of Unix for websites". W3Techs. 7 March 2015.
99. ^ "Web Server Survey". Security Space. 1 March 2014.
100. ^ "OS/Linux Distributions using Apache". Security Space. 1 March 2014.
101. ^ "OS Market Share and Usage Trends". W3Cook.com. Retrieved 30 June 2015.
102. ^ "Web Technologies Statistics and Trends". W3Techs. December 2013.
103. ^ "W3Cook FAQ". W3Cook.com. Retrieved 30 June 2015.

Einführung: Geeigneter Editor

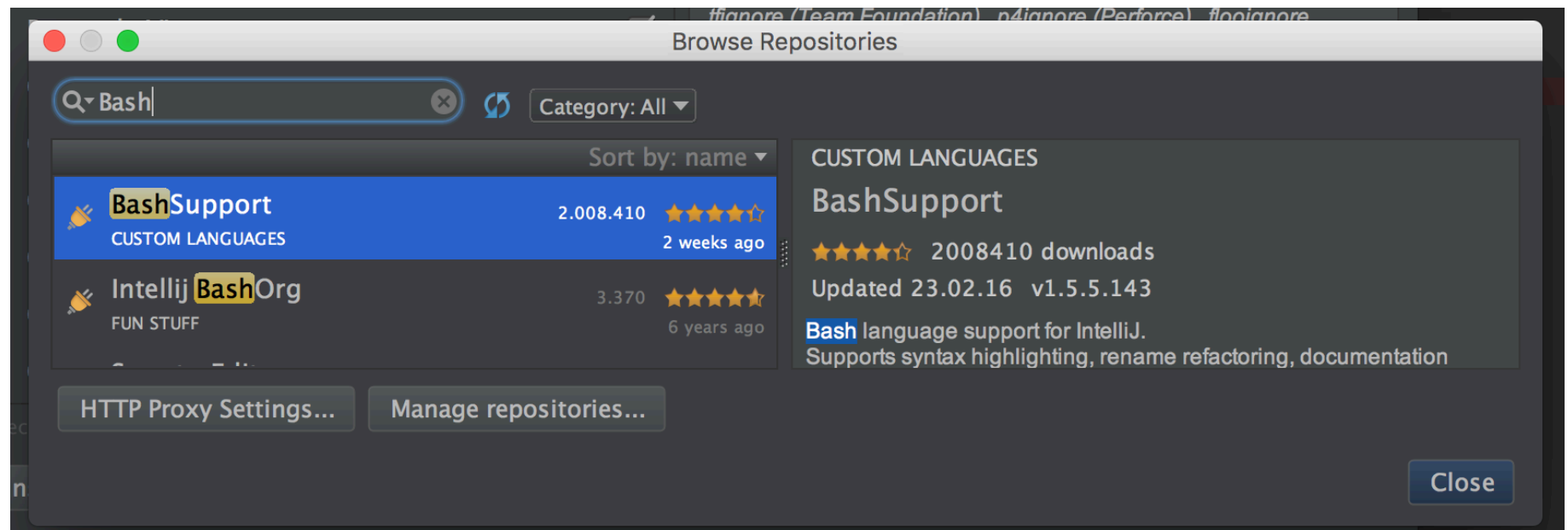
- Non-Interactive

```
→ ~ echo "#! /bin/bash" > hi.sh
→ ~ echo "echo \"hi\"" >> hi.sh
→ ~ cat hi.sh
#!/bin/bash
echo "hi"
```

- Sehr verbreiteter Editor: visual improved, kurz vim
 - :set paste
 - Einstellungen über .vimrc

Einführung: Geeigneter Editor

- IntelliJ Bash Plugin
- Sublime (Syntax Shell Script (Bash))



Einführung: Hello World

- Datei erstellen:

```
touch test.sh; chmod 700 test.sh; vim test.sh
```

- Shebang

Einführung: Begriffe

- Builtins
- Expansions, unter anderem:
 - Parameter Expansion
 - Brace Expansion
 - Tilde Expansion
- Substitutions, unter anderem:
 - Command Substitution
 - Process Substitution

Einführung: Grundlegendes

- \$@, \$*, \$#, \$1,...,\$9, shift vs \${10}, \$?
- Compound Commands
 - Grouping

```
echo "1"; (exit 1); echo "2"  
echo "1"&& (exit 1)&& echo "2"
```

?

Einführung: Conditionals

- Compound Commands: Conditionals

- Test / []

```
test 2 -gt 5; echo $?  
test -e `pwd`/loops.sh; echo $?  
[ -n "content" ];echo $?
```

- (()) Compound Command

- [[]] Compound Command

Einführung: Debugging

- `/bin/bash -xv file`
- `echo „command“; command; exit 0;`
- Debug an/aus

```
#!/bin/bash
echo Hey
set -x
echo Hey
set +x
echo Hey
```

- Weitere Set-Parameter: `set -u`, `set -e`, `set -C`

Einführung: Debugging

- ShellCheck

```
→ introduction cat shellchk.sh
```

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

```
NOTUSED="notused"
```

```
→ introduction shellcheck shellchk.sh
```

In shellchk.sh line 2:

```
NOTUSED="notused"
```

```
^-- SC2034: NOTUSED appears unused. Verify it or export it.
```

Häufige Probleme

- Variablen
 - Erstellung
 - Benennung/Namensänderung
- Zeichenabstände
- Fehlende Keywords
- Pipes und Reihenfolge stdin/out
 - Komplette Umleitung (2>&1) erst nach Schreiben in Datei
<http://stackoverflow.com/questions/4699790/cmd-21-log-vs-cmd-log-21>

Häufige Probleme

- Variablen
 - Erstellung
 - Benennung/Namensänderung
- Beispiel: Einfache Parameter Expansion

```
1 #! /bin/bash
2
3 WORD="word"
4 echo "The plural of $WORD is most likely $WORDS"
```

- ?

```
6 WORD="word"
7 echo "The plural of $WORD is most likely ${WORD}s"
```

Häufige Probleme

- Quotes

```
mylist="DOG CAT BIRD HORSE"
```

```
for animal in "$mylist"; do  
    echo $animal  
done
```

- ?

```
for animal in $mylist; do  
    echo $animal  
done
```

<http://www.davidpashley.com/articles/writing-robust-shell-scripts/>, abgerufen 01.5.16

- \$@, test Arguments
- Regel: Im Zweifel immer Double Quotes um

Häufige Probleme

- Escaping

```
#!/bin/bash
echo 'Here's the path \''
```

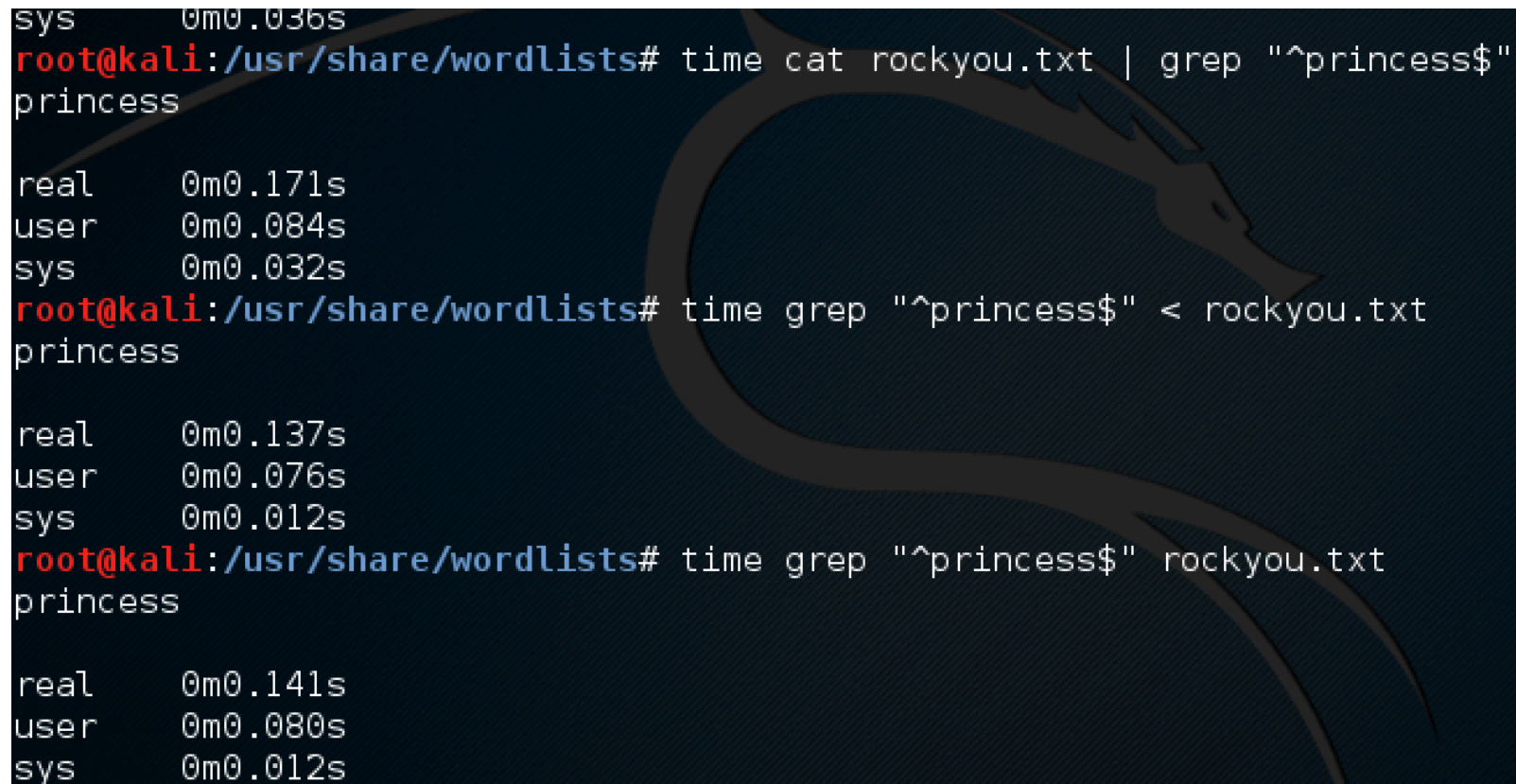
?

```
#!/bin/bash
# Strong quoting
echo 'Here\'\'s the path \''
# Weak quoting
echo "Here's the path \\"
```

```
1 # Per-Character Escaping
2 echo \"$HOME is set to \"$HOME\""
```


Best Practices

- Kommandos auf eigene Parameter/Stdin-Eingaben prüfen



```
sys      0m0.036s
root@kali:/usr/share/wordlists# time cat rockyou.txt | grep "^princess$"
princess

real     0m0.171s
user     0m0.084s
sys      0m0.032s
root@kali:/usr/share/wordlists# time grep "^princess$" < rockyou.txt
princess

real     0m0.137s
user     0m0.076s
sys      0m0.012s
root@kali:/usr/share/wordlists# time grep "^princess$" rockyou.txt
princess

real     0m0.141s
user     0m0.080s
sys      0m0.012s
```

https://wiki.ubuntuusers.de/Shell/Tipps_und_Tricks/, abgerufen 26.4.16

Best Practices

- Naming

```
→ test ls
→ test echo "echo Hello" > test; chmod 700 test
→ test ls
test
→ test test
```


- ?

```
→ test test
→ test ./test
Hello
→ test which test
test: shell built-in command
```

Bad Practices

- eval

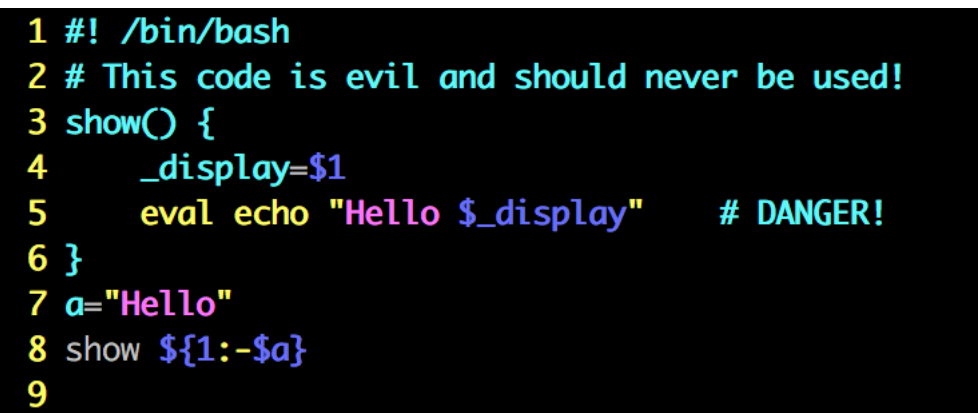
```
1 #!/usr/bin/env bash
2 { myCode=$(
```



A terminal window with a black background and colored text. It shows a script starting with a shebang, opening a brace to read from stdin into a variable, running 'ls', and then using 'eval' to execute the variable's content.

```
3 ls
4 EOF
5
6 eval "$myCode"
```

```
1 #! /bin/bash
2 # This code is evil and should never be used!
3 show() {
4     _display=$1
5     eval echo "Hello $_display"    # DANGER!
6 }
7 a="Hello"
8 show ${1:-$a}
9
```



A terminal window with a black background and colored text. It shows a script with a warning comment, a function 'show' that uses 'eval' to echo a string, and a call to 'show' with a default argument. The comment explicitly states that this code is evil and should never be used.

https://wiki.ubuntuusers.de/Shell/Tipps_und_Tricks/, abgerufen 26.4.16

Ausgewählte Beispiele

- Root Password Phishing
- Performance Boost:
 - tcp connect / udp scan.sh
- Ping Sweep
- Add one wordlist to another: addtowl.sh
- Testdriven Development: test_input.sh

- Weitere Anwendungsbereiche:
 - Brute-Forcing zB Dns (Reverse) Enumeration
 - Transaktionale Programmierung

Zusammenfassung

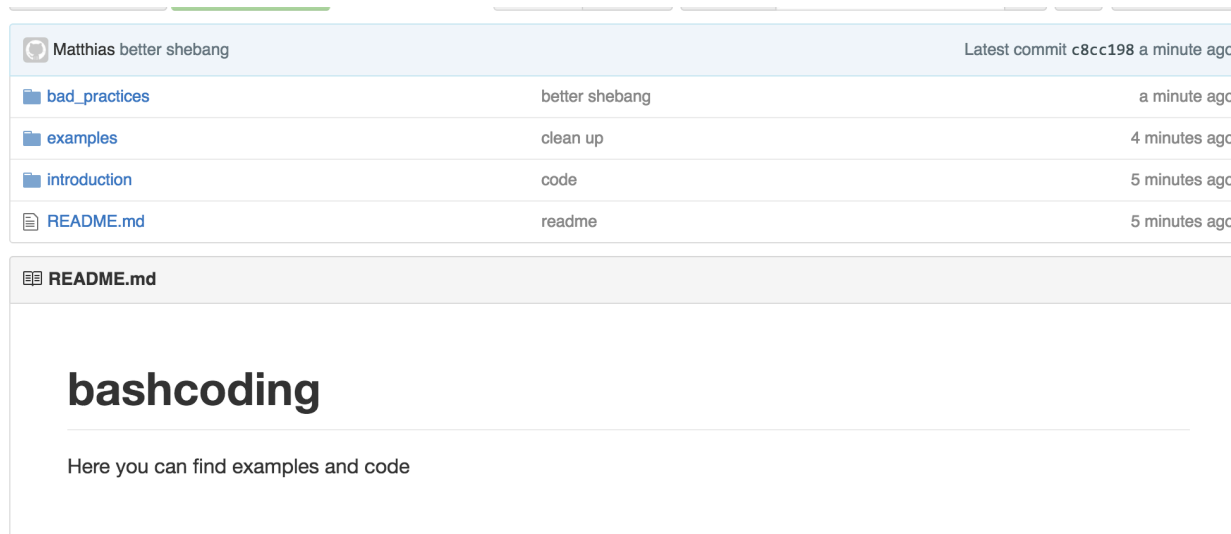
1. Motivation
2. Einführung
3. Häufige Probleme
4. Best Practices / Bad Practices
5. Ausgewählte Beispiele

Code zur Präsentation

<https://github.com/mr4p/bashcoding.git>

bzw

<https://github.com/mr4p/bashcoding>



The screenshot shows the GitHub repository page for 'bashcoding' by user 'Matthias better shebang'. The repository has a latest commit 'c8cc198' made 'a minute ago'. The file list includes:

File/Folder	Commit Message	Time Ago
bad_practices	better shebang	a minute ago
examples	clean up	4 minutes ago
introduction	code	5 minutes ago
README.md	readme	5 minutes ago

Below the file list is the 'README.md' content:

bashcoding

Here you can find examples and code

Weitere Informationen

- Einführung:
 - https://en.wikibooks.org/wiki/Bash_Shell_Scripting
 - Klammern:
<http://stackoverflow.com/questions/2188199/how-to-use-double-or-single-bracket-parentheses-curly-braces>
- Fortgeschrittene:
 - <http://mywiki.woledge.org/BashGuide>
 - <http://wiki.bash-hackers.org/>
 - <http://tldp.org/LDP/abs/html/>
- Best Practices
 - https://wiki.ubuntuusers.de/Shell/Tipps_und_Tricks/
 - <http://www.davidpashley.com/articles/writing-robust-shell-scripts/>
- Offizielle Seiten
 - Gnu Manual: <http://www.gnu.org/software/bash/manual/bash.html>
 - Posix Spec:
http://pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html

Vielen Dank!

Backup - Bash Coding

Security Meetup

Mittwoch, 11. Mai 2016

Matthias Altmann

Motivation

- Häufige Probleme:
 - Wiederholbare Aufgaben Kommandozeile / Aufwand
- Lösung:
Befehle automatisierbar ablegen
- Unix: Verwendung Skriptsprache einer Shell
- Verbreiteste Shell: Bourne Again Shell (Bash)

Motivation

- Weiterer Vorteil Bash-Coding Skills
- Unix Philosophie: DOTADIW = Do One Thing And Do It Well
- Viele kleine (systemnahe) Werkzeuge
- Scriptbefehle immer auch in der Konsole anwendbar
- Produktivität auf der Konsole steigt
- Aus Security-Sicht
 - Möglichkeiten der „Macht“ über ein System steigen

Einführung: Geeigneter Editor

- Sehr verbreiteter Editor: visual improved, kurz vim
- Häufige Befehle:
 - Esc: wq!,q!
 - u
 - I, A
 - dd
 - v,y/c,p
 - gg, G
 - ^,\$
 - :set number
 - :set paste
- Einstellungen über .vimrc

Einführung: Begriffe

- Functions
- Builtins
 - let
 - local

Einführung

- String-Vergleiche

```
#!/bin/bash
output="not empty"
if [ ! -z "$output" ]
then
    echo "$output"
else
    echo nope
fi
```

Einführung

- File Descriptoren / Redirection

```
find . -print | grep -i bla 2> /dev/null
```

- source file
- Backticks / \$()

Einführung: Process Substitution

```
1 diff <(ls dir1) <_(ls dir2)_
```


Einführung: [[]] Compound Command

- Moderne Variante des test Kommandos
- Arithmetische Operationen möglich
- Pattern Matching möglich

```
#!/bin/bash
[[ 'i=5, i+=2' -eq 3+4 ]] && echo true
VARIABLE="Let's test this string!"

# This is for regular expressions:
if [[ "$VARIABLE" =~ L*ing! ]]
then
    echo "matched"
else
    echo "nope"
fi
```

Einführung: Command Substitution

```
1  #!/bin/bash
2
3  function get_password ( )
4  # Usage:      get_password
5  # Asks the user for a password; prints it for capture by calling code.
6  # Returns a non-zero exit status if standard input is not a terminal, or if
7  # standard output *is* a terminal, or if the "read" command returns a non-zero
8  # exit status.
9  {
10     if [[ -t 0 ]] && ! [[ -t 1 ]] ; then
11         local PASSWORD
12         read -r -p 'Password:' -s PASSWORD && echo >&2
13         echo "$PASSWORD"
14     else
15         return 1
16     fi
17 }
18
19 echo "$(get_password)"
20
```

Einführung: Stringhandling / Parameter Expansion

```
1  #!/bin/bash
2  echo ${*:2}           # Echoes second and following positional parameters.
3  echo {@:2}           # Same as above.
4  echo ${*:2:3}         # Echoes three positional parameters, starting at second.
5  stringZ=abcABC123ABcabc
6  echo ${#stringZ}      # 15
7  echo ${stringZ:0}     # abcABC123ABcabc
8  echo ${stringZ:1}     # bcABC123ABcabc
9  echo ${stringZ:7}     # 23ABcabc
10 echo ${stringZ:7:3}   # 23A
11 echo ${stringZ:-4}    # abcABC123ABcabc
12 echo ${stringZ:(-4)}  # cabc
13 echo $stringZ | tr '[:upper:]' '[:lower:]'
14 echo $stringZ | tr '[:lower:]' '[:upper:]'
15
```

Einführung: Parameter Expansion

- Substring Removal

```
1 #! /bin/bash
2
3 FILENAME="test.txt"
4 # Get name without extension
5 echo ${FILENAME%.*}
6 # ⇒ test
7
8 # Get extension
9 echo ${FILENAME##*.}
10 # ⇒ txt
11
12 # Get directory name
13 PATHNAME="/home/bash/bash_hackers.txt"
14 echo ${PATHNAME%/*}
15 # ⇒ bash_hackers.txt
16
17 #Get filename
18 echo ${PATHNAME##*/}
19 #⇒ bash_hackers.txt
20
21
22
```

Häufige Probleme

- Piping und Variablen

```
1 # printf übergibt 2 Zeilen die mittels read gezählt werden
2 Zaehler=0
3 printf "%s\n" foo bar | while read -r line
4 do
5     Zaehler=$((Zaehler+1))
6     echo "Zaehler in der Schleife: $Zaehler" |
7 done
8 echo "Zaehler nach der Schleife: $Zaehler"
```

- Lösung Process Substitution

```
1 #!/bin/bash
2 Zaehler=0
3 while read -r line
4 do
5     Zaehler=$((Zaehler+1))
6     echo "Zaehler in der Schleife: $Zaehler"
7 done <<(printf "%s\n" foo bar)
8 echo "Zaehler nach der Schleife: $Zaehler"
```

Quotes und \$@

```
1 #!/bin/bash
2 foo() { for i in $@; do printf "%s\n" "$i"; done };
```

?

```
3
4 foo2() { for i in "$@"; do printf "%s\n" "$i"; done };
5
```

- Im Zweifel immer Quotes bei \$@ nutzen

<http://www.davidpashley.com/articles/writing-robust-shell-scripts/>, abgerufen 01.5.16

- Das gleiche gilt für test conditional

Best Practices

- Verwendung von set -u

```
1 #!/bin/bash
2 set -u
3 UNUSED=$1
```

- Verwendung von set -e (rm, mkdir -p usw)

```
1 #! /bin/bash
2 # set -e
3 (exit 1)
4 if [ "$?" -ne 0 ]; then echo "command 1 failed"; fi
5
6 set -e
7 echo "Check for non-return value active"
8 if ! (exit 1); then echo "command 2 failed"; fi
9
10 set +e
11 (exit 1)
12 (exit 2)
13 echo "Multiple commands failed"
14 set -e
15
```

<http://www.davidpashley.com/articles/writing-robust-shell-scripts/>, abgerufen 01.5.16

Best Practices

- Sinnloses ls

```
1 for i in $(ls *)
2 do
3     cat $i
4 done
```

```
1 for i in *
2 do
3     cat $i
4 done
```

```
1 for i in *
2 do
3     echo "$i" # Variable in " "
4 done
```

```
1 cat *
```

https://wiki.ubuntuusers.de/Shell/Tipps_und_Tricks/, abgerufen 26.4.16

Best Practices

- Langsames grep

```
3 grep "Tag" tosearch
4 if [ $? -ne 0 ]
5 then
6     echo "not found"
7 fi
```

- Besser

```
10 # grep -q bricht Suche beim ersten Treffer ab
11 if ! grep -q "Tag" tosearch
12 then
13     echo "not found"
14 fi
```

https://wiki.ubuntuusers.de/Shell/Tipps_und_Tricks/, abgerufen 26.4.16