

Kapitel 2: Shell-Programmierung

2: Einführung in die Shell-Programmierung

- ▶ Folgen von Shell-Kommandos nennt man Shellsript
- ▶ Die Shell unterstützt Verzweigungen, Schleifen und Funktionsaufrufe
- ▶ Ein Shell-Skript ist eine Textdatei mit mehreren Shell- bzw. Linux-Kommandos
- ▶ Typische Anwendungen: Automatisierung von Kommandofolgen zur Administration, Durchführung von Backups, ...
- ▶ Hilfe zur Shell-Programmierung: `man bash` oder `man sh`

Starten und Verlassen

- ▶ `$ sh`
Startet eine Shell
- ▶ `$ bash`
Startet eine GNU Bourne-Again SHell
- ▶ `$ exit <nummer>`
 - ▶ Beendet die Shell mit Rückgabewert `<nummer>`
 - ▶ Beim beenden gibt die `main()`-Funktion des Programms `bash` den Wert `<nummer>` zurück.

Hallo Welt

```
$ cat hallo_welt.sh
```

```
#!/bin/sh  
# Dies ist ein Testprogramm  
echo "Hallo_Welt"
```

Kommentare und Startzeile

- ▶ Das Zeichen ' #' leitet einen Kommentar ein.
- ▶ Ein Kommentar endet mit dem Zeilenumbruch.
- ▶ Unter Linux wird in der ersten Zeile eines Skriptes (Startzeile) der Interpreter angegeben.
- ▶ Liste von gängigen Interpretern.
 - ▶ `/bin/sh`
 - ▶ `/bin/bash`
 - ▶ `/usr/bin/perl`
 - ▶ `/usr/bin/python`
- ▶ Die erste Zeile eines Skriptes wird oft Shebang bzw. Hashbang genannt, da dies **immer** mit einem # gefolgt von einem ! beginnt.
- ▶ **Shebang Beispiel:** `#!/bin/sh`

Ausführbarkeit von Skripten

- ▶ Linux kennt drei Zugriffsrechte
 - ▶ Lesen (read, r)
 - ▶ Schreiben (write, w)
 - ▶ Ausführen (execute, x)
- ▶ Neu angelegt Dateien sind normalerweise nicht ausführbar.
- ▶ Mit dem Kommando `chmod +x foo.sh` wird die Datei `foo.sh` ausführbar

```
$ ./hallo_welt
hallo_welt.sh : Keine Berechtigung
```

```
$ chmod +x hallo_welt.sh
```

```
$ ./hallo_welt.sh
Hallo Welt
```

Rückgabewert

Jeder Prozess liefert beim Beenden einen ganzzahligen Rückgabewert (exit code) zurück

- ▶ `exit code == 0`: Prozess wurde erfolgreich beendet
- ▶ `exit code != 0`: Prozess wurde nicht erfolgreich beendet
- ▶ Der Rückgabewert ist in der Variable `$?` gespeichert

Beispiel:

```
$ cat /dev/null; echo $?
```

```
0
```

```
cat fuuu; echo $?
```

```
cat: fuuu: Datei oder Verzeichnis nicht gefunden
```

```
1
```

2.1: Shell-Variablen

- ▶ In der Shell lassen sich auch Variablen anlegen

```
$ var=Hallo
```

```
$ var="Hallo Welt"
```

```
$ var='Hallo Welt'
```

- ▶ Zugriff auf eine Shell-Variable erfolgt mit dem Operator `$`

```
$ echo $var
```

```
Hallo Welt
```

Achtung

Links und rechts vom Zuweisungsoperator (=) darf KEIN Leerzeichen vorkommen.

Spaß mit Zuweisungen

```
var=Hallo
```

Der Variable `var` wird der String `Hallo` zugewiesen.

```
var =Hallo
```

Das Programm `var` wird mit dem Parameter `=Hallo` aufgerufen.

```
var = Hallo
```

Das Programm `var` wird mit dem Parametern `=` und `Hallo` aufgerufen.

```
var= Hallo
```

Vordefinierte Shell-Variablen

`?`: Rückgabewert des letzten Kommandos

`!`: PID des zu letzt gestarteten Hintergrundprozess

`$`: PID der aktuellen Shell

`0`: Dateiname des gerade ausgeführten Shellskriptes

`#`: Anzahl der übergebenen Parameter

`*`: Übergabeparameter als zusammenhängender String

`@`: Übergabeparameter als Folge von Strings

`1` bis `9`: Übergabeparameter 1 bis 9

Beispiel

```
$ cat parameter.sh
```

```
#!/bin/bash  
echo $#  
echo $*  
echo $0  
echo $2
```

```
$ ./parameter.sh Hallo Welt  
...  
$ ./parameter.sh Hallo  
...  
$ ./parameter.sh  
...
```

Frage: Was gibt das obige Shellskript?

Spaß mit Ganzzahlige Variablen

▶ Variablenzugriff vs. String

- ▶ Lesender Zugriff auf die Variable `Hallo`:

```
$Hallo
```

- ▶ Ausgabe des Strings `Hallo`:

```
$ echo -n Hallo
```

- ▶ Ausgabe des Wertes der Variable `Hallo`:

```
$ echo -n $Hallo
```

- ▶ Bei arithmetischen Operationen gibt es keine Doppeldeutigkeit
⇒ Präfix **\$** wird bei arithmetischen Operationen nicht benötigt

- ▶ Bsp.: `$ x=5; echo $((x * x))`
25

Spaß mit Strings

Regel

Strings immer in doppelte Anführungszeichen setzen, ansonsten werden oftmals Sonderzeichen wie `*` oder `?` als Seiteneffekt ausgewertet.

```
#!/bin/sh
```

```
a=" *
```

```
echo $a
```

```
echo "$a"
```

Frage: Was ist die Ausgabe des Shellskriptes?

Umgebungsvariablen

- ▶ In der Shell wird zwischen lokalen und globalen Variablen (Umgebungsvariable) unterschieden
- ▶ **Environment**: Menge der gesetzten Umgebungsvariablen die dem Betriebssystem bekannt gemacht wurden
- ▶ Umgebungsvariable werden der Shell oder einem Prozess übergeben; lokale Variablen nicht
- ▶ Kindprozesse haben Zugriff auf die **Environment**
- ▶ Die Shell ist der Vaterprozess aller von dort aufgerufenen Programme
- ▶ Zur **Environment** gehört z.B. die Systemsprache, Proxyeinstellungen sowie eine Reihe verschiedener Pfade
- ▶ Das Kommando `printenv` gibt die **Environment** aus
- ▶ Das Kommando `set` zeigt **alle** Variablen an

Wichtige Umgebungsvariablen

- ▶ HOME: Homeverzeichnis (**Merke**: ~ steht für \$HOME)
- ▶ HOSTNAME: Rechnername
- ▶ PS1: Shell-Prompt
- ▶ PATH: Suchpfad für ausführbaren Programme
- ▶ PWD: Aktuelles Verzeichnis
- ▶ OLDPWD : Vorheriges Verzeichnis

Setzen, Modifizieren und Löschen von Variablen

Beispiele für die Variable `FOO`, den Wert `val` und dem Kommando `ls`.

- ▶ `FOO=val`
Weist der Variable einen Wert zu
- ▶ `FOO=$(ls)` oder `FOO=`ls``
Weist der Variablen die `stdout` des Kommandos zu
- ▶ `FOO=$FOOue`
Anfügen des Strings `"ue"`
- ▶ `export FOO`
Fügt eine Variable der **Environment** hinzu
- ▶ `unset FOO`
Löscht eine (Umgebungs)variable

Beispiel

```
FOO=42
FOO=$FOO.23
echo $FOO

BAR="date_+%T"
echo $BAR

BAR=`date +%T`
echo $BAR

set | grep BAR
env | grep BAR

export BAR
env | grep BAR
unset BAR
echo $BAR
```

Variablen einlesen

- ▶ Manche Shell-Skripte benötigen Benutzerinteraktion:
 - ▶ Zustimmung zu Lizenzbestimmungen
 - ▶ Ausfüllen eines Formulars zur Generierung einer pdf-Datei
 - ▶ Nachfrage ob eine Operation wirklich ausgeführt werden soll
 - ▶ ...
- ▶ Das Kommando `read` erlaubt das Einlesen von Benutzereingaben
- ▶ `$ read <variable>`
Zuweisung einer Variable mittels Benutzereingabe.

Beispiel: (greeter.sh)

```
#!/bin/sh
echo -n "What's_your_name?"
read name
echo Hi $name. Nice to meet you.
```

Kommandosubstitution

- ▶ Kommandosubstitution bezeichnet das Ersetzen eines Kommandos durch dessen Ausgabe
- ▶ In der Shell findet die Kommandosubstitution durch `$ (cmd)` oder Backticks (``cmd``) statt
- ▶ `$ d=$(date)`: Der Variable `d` wird die Standardausgabe des Kommandos `date` zugewiesen
- ▶ Kommandosubstitution kann beliebig verschachtelt werden
Beispiel: `foo=$(basename $(pwd))`

Kommandosubstitution und Whitespaces

- ▶ **Achtung:** Bei der normalen Ausgabe eines Variableninhaltes werden alle Zwischenraumzeichen (engl. *white space*) – wie Leerzeichen, Tabulator, oder Zeilenumbruch – durch Leerzeichen ersetzt.

Beispiel: Die Ausgabe von `d=$(cal | head -n 3); echo $d` ist

```
Juni 2016 So Mo Di Mi Do Fr Sa  1 2 3 4
```

- ▶ Die **doppelten Anführungszeichen** verhindern die Ersetzung von Whitespaces

```
$ d=$(cal | head -n 4); echo "$d"
Juni 2016
So Mo Di Mi Do Fr Sa
      1  2  3  4
5   6  7  8  9 10 11
```

Sonderzeichen in Zeichenketten

- ▶ Sonderzeichen haben in der Shell meist eine besondere Bedeutung
- ▶ Daraus ergeben sich praktische Probleme
- ▶ Wie greife ich auf eine Datei `$PWD` oder `ls | wc -l` zu?
- ▶ Mittels `\` lassen sich Sonderzeichen escapen
- ▶ Mittels `'<string>'` lässt sich eine ganze Zeichenkette escapen
- ▶ `echo HALLO WELT > '$PWD'`
Legt die Datei `$PWD` mit dem Inhalt `HALLO WELT` an
- ▶ Was passiert hier?
 - ▶ `echo $PWD`
 - ▶ `cat '$PWD'`
 - ▶ `cat \"$PWD`

2.2: Kontrollstrukturen

Agenda

- ▶ `if-else`-Verzweigungen
- ▶ `case`-Verzweigungen
- ▶ Die `for`-Schleife
- ▶ Die `while`-Schleife

if-else-Verzweigung

Syntax

```
if <condition>; then
...
else  # Der else-Zweig ist optional
...
fi
```

Eine Bedingung (engl. condition) ist erfüllt falls...

- ▶ ...ein Programm erfolgreich beendet wurde.

Beispiel: `if true; then`

- ▶ ...der Ausdruck [<condition>] wahr ist.

Beispiel: `if [$# -ne 2]; then`

- ▶ Pro Tip für die Syntax von if-conditions: **man test**

if-Bedingung: Strings

- ▶ `[-n s1]`: Teste ob `s1` kein leerer String ist.
- ▶ `[-z s1]`: Teste ob `s1` **der** leerer String ist.
- ▶ `[s1 = s2]`: Teste ob `s1` und `s2` identisch sind.
- ▶ `[s1 != s2]`: Teste ob `s1` und `s2` ungleich sind.

if-Bedingung: Ganzzahlen

- ▶ `[i -eq j]`: Test ob $i = j$ gilt.
- ▶ `[i -ge j]`: Test ob $i \geq j$ gilt.
- ▶ `[i -gt j]`: Test ob $i > j$ gilt.
- ▶ `[i -le j]`: Test ob $i \leq j$ gilt.
- ▶ `[i -lt j]`: Test ob $i < j$ gilt.
- ▶ `[i -ne j]`: Test ob $i \neq j$ gilt.

if-Bedingung: Dateien

- ▶ `[f1 -ot f2]`: Test ob Datei `f1` älter als Datei `f2` ist.
- ▶ `[-e file]`: Test ob `file` existiert.
- ▶ `[-f file]`: Test ob `file` eine reguläre Datei ist.
- ▶ `[-d file]`: Test ob `file` ein Verzeichnis ist.
- ▶ `[-h file]`: Test ob `file` ein symbolischer Link ist.
- ▶ `[-s file]`: Test ob `file` leer ist.
- ▶ `[-r file]`: Test ob `file` lesbar ist.
- ▶ `[-w file]`: Test ob `file` schreibbar ist.
- ▶ `[-x file]`: Test ob `file` ausführbar ist.

Formulierung von Bedingungen mit `test`

- ▶ Angabe von Bedingungen ist in der Shell limitiert
- ▶ Sonderzeichen wie `>` sind bereits vergeben
- ▶ Lösung: Das Kommando `test`
- ▶ `test` gibt 0 zurück falls eine Bedingung erfüllt ist, ansonsten 1.
- ▶ `test $X`: Tested ob die Variable `X` belegt ist
- ▶ `test $X -gt 5`: Tested ob die Variable einen Zahlenwert größer 5 enthält
- ▶ `test "$X" = "FOO"`
Tested ob `X` mit dem Wert `"FOO"` belegt ist.
- ▶ `test <condition>` entspricht `[<condition>]`
- ▶ Weitere Infos gibt es in der `test`-Manpage und unter <http://wiki.bash-hackers.org/commands/classictest>

Beispiel: iftwo.sh

```
#!/bin/sh
if [ $# -ne 2 ]; then
    echo "This_command_requires_exactly_2_arguments"
else
    echo "Argument_1:_$1,_Argument_2:_$2"
fi
```

Frage: Was ist die Ausgabe des Skripts?

- ▶ `$ iftwo.sh true`
- ▶ `$ iftwo.sh Hallo welt`
- ▶ `$ iftwo.sh Dies ist ein Test`

Beispiel: if_exec_test.sh

```
#!/bin/sh

if $* >/dev/null 2>&1; then
    echo Command \"$*\" succeeded
else
    echo Command \"$*\" failed
fi
```

Frage: Was ist die Ausgabe des Skripts?

- ▶ \$ if_exec_test.sh true
- ▶ \$ if_exec_test.sh echo Hallo
- ▶ \$ if_exec_test.sh false

Die `elif`-Anweisung

`elif` ist die Kurzschreibweise für `else if`.

Beispiel:

```
#!/bin/sh
if [ -d "$1" ]; then
    echo "$1" ist ein Verzeichnis
elif [ -f "$1" ]; then
    echo "$1" ist eine Datei
else
    echo "$1" ist eine spezielle Datei
fi
```

Case-Verzweigungen

Syntax

```
case "$x" in
  foo|bar) ...;;
  *.txt) ...;;
  ...
  *) ...;; # Default case
esac
```

- ▶ Es werden der Reihe nach alle Fälle getestet
- ▶ Nur der erste passende Fall wird abgearbeitet
(*Wer zuerst kommt, mahlt zuerst*-Prinzip)
- ▶ Ein `case`-Zweig wird mit einem Doppelsemikolon (; ;) beendet

Beispiel casedemo.sh

```
#!/bin/sh

case "$1" in
  *.tex) echo "LaTeX-Datei";;
  *.t*)  echo "Textdatei";;
  *.jpg | *.png) echo "Bilddatei";;
  *.ps   | *.pdf) echo "Dokument";;
  *.avi  | *.wmv) echo "Videodatei";;
  *.mp3  | *.ogg) echo "Musikdatei";;
  *)     echo "Unbekannter_Dateityp";;
esac
```

Frage: Was ist die Ausgabe des Skripts?

- ▶ `./casedemo.sh foo.mp3`
- ▶ `./casedemo.sh foo.iso`
- ▶ `./casedemo.sh foo.tex`

Die `for`-Schleife

Syntax

```
for element in list; do
    Befehl1
    Befehl2
    ...
done
```

Die `for`-Schleife wird für jedes Element aus der Liste `list` aufgerufen

Beispiele für Mengen

- ▶ `a b c`
- ▶ `$(ls)`

Beispiel `fordemo`

```
#!/bin/sh
counter=1
for filename in $(ls *.sh); do
    echo $counter. $filename
    counter=$((counter + 1))
done
```

Frage: Was macht das Skript?

Frage: Können wir `$((counter + 1))` durch `[$counter +1]` ersetzen?

Die `while`-Schleife

Die `while`-Schleife wird durchlaufen solange die Schleifenbedingung erfüllt ist.

```
#!/bin/sh
i=0
while [ $i -lt 10 ]; do
    echo $i
    i=$(( i + 1 ))
done
```

Frage: Was macht das Skript?

2.3: Funktionen

- ▶ Ein Shell-Skript kann auch Funktionen enthalten
- ▶ Auf die übergebenen Argumente kann mittels `$#`, `$*`, `$1`, `$2`, ... zugegriffen werden.
- ▶ Variablen, die in einer Funktion angelegt werden, sind nur lokal gültig
- ▶ Mit der Returnanweisung kann die Funktion eine Zahl zurückgeben. Beispiel: `return 42`
- ▶ Rückgabewert kann mittels `$?` erfragt werden

Beispiel: Summe I

```
#!/bin/sh

sum() {
    sum=0
    for i in "$@"; do
        sum=$(( sum + i ))
    done
    return $sum
}

sum "$@"
echo $?
```

Beispiel: Summe II

```
#!/bin/sh
sum=0

summe() {
    sum=0
    for i in "$@"; do
        sum=$(( sum + i ))
    done
}

summe "$@"
echo $sum
```

Usage

Jedes Shell-Skript sollte über eine Funktion `usage` verfügen, die...

- ▶ ...dem Benutzer erklärt, wie das Skript aufzurufen ist und
- ▶ anschließend das Shellskript mit dem Befehl `exit 1` beendet.

Beispiel:

```
usage() {  
    echo "Usage: _$0_<filename>" > /dev/stderr  
    exit 1  
}
```

Usage Beispiel

```
#!/bin/sh

usage() {
    echo "Usage: _$0_<filename>" > /dev/stderr
    exit 1
}

# main
if [ ! -e "$1" -o $# -ne 1 ]; then
    usage
fi

if [ -d "$1" ]; then
    echo "$1" ist ein Verzeichnis
elif [ -f "$1" ]; then
    echo "$1" ist eine normale Datei
else
    echo "$1" ist eine spezielle Datei
fi
```


Bilderkonvertierung I

```
1  #!/bin/sh
2  USAGE() {
3      echo "convert.sh <directory> <format>"
4      echo "convert_all_jpg_and_gif_images_to <format>"
5      exit $1
6  }
7
8  CONVERT() {
9      for i in $1/*. $2; do
10         BASENAME=`basename $i | cut -d "." -f 1`
11         convert $i $3:$BASENAME.$3
12     done
13 }
```

Bilderkonvertierung II

```
15 # Main
16 if [ "$#" -ne "2" ]; then
17     USAGE 1
18 fi
19
20 if [ ! -d "$1" ]; then
21     USAGE 2
22 fi
23
24 CONVERT $1 "jpg" "$2"
25 CONVERT $1 "gif" "$2"
```

Caches leeren

Linux cacht Prozesse und Dateien im Speicher (siehe `$ free`). Dies erhöht die Ladegeschwindigkeit bei erneutem Zugriff. Das folgende Skript leert den Cache.

```
#!/bin/sh
free_pagecache=1
free_dentries_and_inodes=2
free_pagecache_dentries_and_inodes=3
ACTION=

USAGE() {
    echo usage: "flush_cache.sh_[1|2|3]"
    exit 1
}

if [ $# -ne 1 ]; then
    USAGE
fi

case "$1" in
    1) ACTION=$free_pagecache;;
    2) ACTION=$free_dentries_and_inodes;;
    3) ACTION=$free_pagecache_dentries_and_inodes;;
    *) USAGE;;
esac
sync; echo $ACTION > /proc/sys/vm/drop_caches
```

Literatur

- Wikibook: Shellprogrammierung
http://de.wikibooks.org/wiki/Linux-Kompendium:_Shellprogrammierung
- Einführung in die Shell-Programmierung von Pawel Slabiak <http://www.linux-services.org/shell/>
- Shell-Programmierung von Jürgen Wolf
http://openbook.rheinwerk-verlag.de/shell_programmierung/
- Das Bash Hackers Wiki
<http://wiki.bash-hackers.org>
- Lecture 04: Unix Shell von Kenneth M. Anderson
<https://www.cs.colorado.edu/~kena/classes/3308/f06/lectures/04/>

Zusammenfassung

Sie sollten ...

- ▶ ... in der Lage sein Shellskripte zu starten.
- ▶ ... in der Lage sein Shellskripte zu schreiben.
- ▶ ... wissen was eine Umgebungsvariable ist.
- ▶ ... in einem Shellskript mit Ganzzahlen rechnen können.
- ▶ ... die vordefinierten Shell-Variablen kennen.
- ▶ ... mit Sonderzeichen in Zeichenketten umgehen können.
- ▶ ... die Kontrollstrukturen kennen.
- ▶ ... Fehler in einem Shellskript beheben können.