unix / linux

```
#!/bin/sh -e
### BEGIN INIT INFO
# Provides:          ifupdown
# Required-Start:    ifupdown-clean
# Required-Stop:     $local_fs
# Default-Start:     S
# Default-Stop:      0 6
# Short-Description: Prepare the system for taking up
### END INIT INFO

[ -x /sbin/ifup ] || exit 0
[ -x /sbin/ifdown ] || exit 0

. /lib/lsb/init-functions

MYNAME="${0##*/}"
report() { echo "${MYNAME}: $*" ; }
report_err() { log_failure_msg "$*" ; }
```

Die Shell als Programmiersprache

Ziele

- Die Programmiersprachen-Eigenschaften der Shell (Variablen, Kontrollstrukturen, Funktionen) kennen
- Einfache Shellskripte erstellen können, die diese Eigenschaften ausnutzen

Agenda

- **Variablen**
- Kontrollstrukturen

Grundlagen

- Keine Deklaration nötig
- Keine Leerzeichen zwischen Name und Werte

```
farbe=blau
```

- Substitution mittels `$`

```
> echo $farbe
blau
```

  - `\$` und `''` verhindern Substitution

Umgebungsvariablen

- "Normale" Variablen sind nur im Kontext der Shell gültig
- Variablen welche auch Kindern zugänglich gemacht werden sollen müssen exportiert werden

```
export farbe
export farbe=blau
```

- Exportierte Variablen können auch wieder gelöscht werden

```
export -n farbe
```

Wertzuweisung

- Komplizierter Zuweisung
  `PATH=$PATH:$HOME/bin`
- Ausgabe eines Kommandos
  `dir=`pwd`` oder `dir=$(pwd)`
- Variable "löschen"
  `var=""` oder `unset var`

Spezielle Shellvariable

`$?`	Rückgabewert des letzen Kommandos
`$$`	Prozessnummer (PID) der aktuellen Shell
`$!`	PID des zuletzt gestarteten Hintergrundprozesses
`$0`	Name des gerade ausgeführten Shell-Skripts
`$#`	Anzahl der Parameter
`$*`	Alle Parameter (“`$1 $2 $3`“)
`$@`	Alle Parameter (“`$1`“ “`$2`“ “`$3`“)
`$`*n*	n-ter Paramter

Standardwerte

- Zuweisen und substituieren:
  - falls sie keinen Wert oder '' hat
    `${farbe:=gelb}`
  - falls keinen Wert hat
    `${farbe=gelb}`
- Substituieren, falls sie keinen Wert oder '' hat
  `${farbe:-gelb}`

Fehlermeldung

```
${name:?<Fehlermeldung>}
```

- Gibt eine Fehlermeldung aus, wenn `name` keinen von '' verschiedenen Wert hat
- Ausführung des Skriptes wird abgebrochen

Teilzeichenketten

- `${name:p:l}`
  - Schneidet aus `name` ab `p l` Zeichen heraus
- `${name#<Muster>}`
  - Entfernt `<Muster>` vom Anfang von `name`
  - `#` entfernt sowenige Zeichen wie möglich
  - `##` entfernt soviele Zeichen wie möglich
- `${name%<Muster>}`
  - Entfernt `<Muster>` vom Ende von `name`

Arithmetische Ausdrücke

- Bash hat primitive Rechenfähigkeiten
  - Nur Ganzzahlen
  - Überlauf wird nicht überprüft
- Benutzt über arithmetische Expansion
  - `a=5; echo $((1+2*a))`

Agenda

- Variablen
- **Kontrollstrukturen**

Rückgabewerte als Steuergrösse

- Rückgabewerte über `$?` auswertbar
- 0 gilt als "erfolgreich", true
- Alle von 0 verschieden Werte gelten als "nicht erfolgreich", false
- Um Schleifen und Bedienungen zu steuern muss auf ein entsprechendes Programm zurückgegriffen werden → `test`

test

- Überprüft
  - Zahlen
  - Zeichenketten
  - Dateieigenschaften

test Beispiele

```
test "$x"          → ist x nicht leer
test $x -gt 7      → ist x grösser als 7
test "$x" \> 10    → ist x lexographisch nach 10
test -r "$x"       → Existiert x und ist lesbar
[ -r "$x" ]        → Kurzform
```

Bedingte Ausführung

- Tue B nur wenn A geklappt hat
  ```
  test -e /etc/default/myprog &&
  source /etc/default/myprog
  ```
- Tue B nur wenn A *nicht* geklappt hat
  ```
  test -d "$HOME/.mydir" || mkdir
  "$HOME/.mydir"
  ```

Einfache Verzweigung

```
if <Testkommando>
then
    <Kommandos bei "Erfolg" (0)>
[elif <Testkommando>
then
    <Kommand bei "Erfolg" (0)>]
[else
    <Kommandos bei "Misserfolg" (1)>]
fi
```

Mehrfachauswahl

```
case <Wert> in
    <Muster1>)
        <Kommandos bei "Muster1">
        ;;
    <Muster2>)
        <Kommandos bei "Muster2">
        ;;
    *)
        <Kommandos bei "Default">
        ;;
esac
```

for Schleife

```
for <Variable> in <Liste>
do
    <Kommandos>
done
```

while Schleife

- Solange Bedingung true (gleich 0) ist

```
while <Test-Kommando>
do
    <Kommandos>
done
```

until Schleife

- Solange Bedingung false (ungleich 0) ist

```
until <Test-Kommando>
do
    <Kommandos>
done
```

Schleifenbehandlung

- Aus der Schleife ausbrechen
  `break`
- Schleifendurchgang abbrechen
  `continue`

Ausnahmebehandlung

- Mit dem Kommando `trap` kann auf Signale reagiert werden

```
trap <Kommando> <Signal(e)>
```

- Es kann auch auf andere Ereignisse wie z.B. `EXIT` reagiert werden.

Funktionen

- Implementierung auch mit "Untershellskripten" möglich
- Innerhalb von Funktionen gelten die Parametervariablen für diejenigen der Funktion
- Funktionen arbeiten wie normale Kommandos (stdin, stdout)

```
function <Name>() {
    <Kommandos>
}
```

Fragen?