Bash

```
echo "Linuxnijmegen #$(seq 2 4 30 | paste -s -d+ | bc) "
```

echo "O, gaat dat zo..!"

Inleiding

We gaan kijken naar:

- waarom dit praatje?
- Quotes
- exit status en true/false
- Conditionele Vergelijkingen
- Rekenkundige expressies
- VVV

Quotes

'verbergt alles voor de shell'

"verbergt alles voor de shell behalve $, \ en `"

bash evalueert altijd (vóór uitvoer van commando's) van links naar rechts

~$ find /usr/bin -name v*

~$ sudo ls > /files

exit status : ZERO = TRUE

exit status 0 -> true

exit status !0 -> false

grep -q oscar /etc/passwd

returns 0 als match -> TRUE

if sudo find /etc -name Passwd -> ??

Conditionele Vergelijkingen 1/3

man: CONDITIONAL EXPRESSIONS

tools: test, [, en [[

- test, [: shell built-ins (voor PATH): **commando's!**
- [[: shell **keyword** (onderdeel vd syntax)

Wat te testen, vergelijken:

- test file attributes
- strings, integer en conditionele vergelijkingen.

[[Conditionele Vergelijkingen 2/3]]

[[: bash extensie (geen POSIX) van [(of test

1. is syntax (bash *keyword*): handelt bv. lege strings:
   - `[ -e $file]` -> ERROR als file leeg is
   - `[ -e "$file" ]` of [[-e $file]]
2. >,<,&& en || kun je gebruiken met [[

   FOUT: `[ -e "$file1"  || -e "$file2" ] &&  echo ok`

   GOED: `[[ -e "$file"  || "A" > "a" ]] &&  echo ok`

   reden: test en [zijn reguliere commando's en >,<,&& en || worden niet meegegeven als arg.

3. **=~** -> reguliere expressies

   ```
   if [[ $answ =~ ^y(es)?$ ]]

   if [ "$answ" = y -o "$answ" = yes ]
   ```

4. globbing: if [["$answ" = y*]]

Conditionele Vergelijkingen 3/3

```
ik=’Harrie Nak’

naam=’Harrie Nak’

[ $naam = $ik ] -> ??

[[ $naam = $ik ]] -> ??

[[ Ik ben $naam = Ik ben $ik ]] -> ??
```

Strategie: wil je POSIX compliant zijn?

Gebruik dan […] (of test …)

Anders: [[…]]

Rekenkundige expressies

man: ARITHMETIC EVALUATION

((<EXPRESSIE>))

exit status:

- Als expressie -> !0 (“arithmetic true"), it returns = 0 (shell true)
- Als expressie -> 0 (“arithmetic false"), returns = 1 (shell false)

`((2+2))` -> ??

`((2-2))` -> ??

Gebruik je meer om te rekenen en niet om te vergelijken:

```
v =$((nbytes/1000))kB
```

```
((i++))
```

rekenkundige expressies

`$((34 / 7))` = POSIX manier

(geen expr en ` meer gebruiken)

VVV 1/3

Zie zeker ook: https://mywiki.wooledge.org/BashPitfalls

```
[ $foo = "bar" ]
```

```
test a \> A (ascii) -> ??
```

```
[[ a > A ]] (lexicografisch) -> ??
```

```
[[ $num > 7 ]]
```
String vergelijking…

```
[ "$num" -gt 7 ] ??
```

```
$((num > 7)) ??
```

VVV 2/3

```
function foo() {
  …
}
echo "~/subdir" ??
echo ~"/subdir" ??
echo ~/"subdir" ??
echo "$HOME/subdir" ??
```

[[$foo = $bar]] ??

Strings vergelijken: quote!

VVV 3/3

filenaam=’bla bla’

printf '%s\n' "Matches met foo: $(grep foo $filename)"

Binnen $(..) quotes **onafhankelijk** van de rest

printf '%s\n' "Matches met foo: $(grep foo "$filename")"

—

count=0

```
grep '/bin/bash' /etc/passwd | while read; do ((count++)); done

while read; do ((count++)); done < <(grep '/bin/bash' /etc/passwd)
```

Dan nog even dit

bash is leuk maar voor echt programmeren: kies een echte programmeertaal.

python is ook “interpreted" maar, naast veel uitgebreider, bv ook veel sneller:

tijd.bash vs tijd.py of tijd.php of tijd.c

(maar denk ik geen hele representatieve test..)

Referentie

$ man bash

https://mywiki.wooledge.org/BashGuide

Bier?