Scripting

Basis van scripting opfrissen

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Dep. PXL-IT – Elfde-Liniestraat 26 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Een script aanmaken

- Aanmaken van een nieuw script
  - doe je best in een nieuwe map “bin” van je homedir
    - cd

      mkdir bin

      cd bin

      vi <scriptnaam> OF nano <scriptnaam>
  - een script kan je de extensie “.sh” geven

Een script aanmaken

- Interpreter
  - geeft aan wat er moet worden gebruikt om de commando's te verstaan (interpreteren)
  - wordt ook dikwijls de shell genoemd
  - wordt gespecifieerd op de eerste regel van het script
  - she-bang (eerste twee tekens)
    - #!/bin/bash --
      - De laatste twee min-tekens kunnen worden geplaatst uit

Een script aanmaken

- Commentaar
  - regel die start met een #-teken
    - # Auteur: Gert Van Waeyenberg
    - # Datum: 20 oktober 2014
    - # Versie: 1.0
    - # Gebruik: ./script.sh <parameter:getal>
  - of ergens in de regel
    - vanaf het #-teken begint de commentaar
      - vDatum=2014   #plaats in deze variabele het huidig jaartal

Een script aanmaken

- Uitvoerbaar maken
  - een script moet uitvoerbaar gemaakt worden
  - als je een zelfgemaakt script uitvoerbaar wilt maken voor jezelf, kan je dit met volgend commando:
    - chmod u+x <scriptnaam>

Input vragen

- Input vragen
  - tijdens de uitvoer van een script kan er naar input gevraagd worden
    - We stellen eerst de vraag
      - echo -n “Geef een getal:”
        - de optie -n zorgt er voor dat de cursor achter de vraag blijft staan
    - Dan vragen we een waarde en kennen deze toe aan een variabele
      - read vGetal
        - Deze variabele kunnen we verder gebruiken in het script

Input vragen - voorbeeld

vwg@laptop: ~/bin

```
#!/bin/bash --

# Auteur: Gert Van Waeyenberg
# Datum:  20 oktober 2014
# Versie: 1.1

echo -n "Geef een getal:"
read vGetal
echo "Het getal dat u gaf was: $vGetal"

~
:wq
```

vwg@laptop: ~/bin

```
vwg@laptop:~/bin$ ./voorbeeldscript.sh
Geef een getal:19
Het getal dat u gaf was: 19
vwg@laptop:~/bin$
```

Het test-commando

- test-commando
  - wordt gebruikt om waarden te vergelijken
    - test $vGetal -gt 100 && echo “Groter” || echo “Kleiner”
      - Indien de inhoud van de variabele groter is dan 100 wordt de tekst “Groter” getoond, anders wordt de tekst “Kleiner” getoond
    - [$vGetal -gt 100] && echo “Groter” || echo “Kleiner”
      - Dit is de verkorte schrijfwijze van het test-commando
      - Let wel op de spatie die verplicht is aan de binnenkant van de vierkante haken

Het test-commando

- test-commando
  - operatoren
    - **-lt:less than**
    - **-gt:greater than**
    - **-ge:greather or equal to**
    - **-le:less or equal to**
    - **-eq:equal to**
    - **-ne:not equal to**

    - **= :equals a string**
    - **!= :Not equals a string**

    - **-d:does dir exist**
    - **-f:does file exist**

Het test-commando - voorbeeld

vwg@laptop: ~/bin

```
#!/bin/bash --

# Auteur: Gert Van Waeyenberg
# Datum:  20 oktober 2014
# Versie: 1.1

echo -n "Geef een getal van 1 tot 20:"
read vGetal
echo "Het getal dat u gaf was: $vGetal"

test $vGetal -lt 10 && echo "Het getal is kleiner dan 10" || echo "Het getal is groter dan 10"

[ $vGetal -lt 20 ] && echo "Het getal is kleiner dan 20" || echo "Het getal is te groot!"

~
~
"voorbeeldscript.sh" 14L, 366C                                1,1           All
```

vwg@laptop: ~/bin

```
vwg@laptop:~/bin$ ./voorbeeldscript.sh
Geef een getal van 1 tot 20:16
Het getal dat u gaf was: 16
Het getal is groter dan 10
Het getal is kleiner dan 20
vwg@laptop:~/bin$
```

Het test-commando

- test-commando
  - Testen op meerdere expressies tegelijk
    - -a voor de AND-operator
    - -o voor de OR-operator

Het test-commando - voorbeeld

vwg@laptop: ~/bin

```
#!/bin/bash --

# Auteur: Gert Van Waeyenberg
# Datum:  20 oktober 2014
# Versie: 1.3

echo -n "Geef een getal van 1 tot 20:"
read vGetal

[ $vGetal -lt 0 -o $vGetal -gt 20 ] && echo "U gaf geen getal van 1 tot 20" || echo "Het getal da
t u gaf is: $vGetal"

                                                                    1,1           All
```

vwg@laptop: ~/bin

```
vwg@laptop:~/bin$ ./voorbeeldscript.sh
Geef een getal van 1 tot 20:-3
U gaf geen getal van 1 tot 20
vwg@laptop:~/bin$ ./voorbeeldscript.sh
Geef een getal van 1 tot 20:15
Het getal dat u gaf is: 15
vwg@laptop:~/bin$ ./voorbeeldscript.sh
Geef een getal van 1 tot 20:27
U gaf geen getal van 1 tot 20
vwg@laptop:~/bin$
```

if-then-else

- if-then-else
  - in plaats van met && en || te werken is het veel duidelijker om te werken met if-then-else
  - de if-then-else-structuur wordt samen gebruikt met het test-commando of zijn verkorte schrijfwijze
  - Opgelet: deze structuur wordt afgesloten met “fi”
    - Dus:

```
if
then
else
fi
```

if-then-else - voorbeeld

vwg@laptop: ~/bin

```
#!/bin/bash --

# Auteur: Gert Van Waeyenberg
# Datum:  20 oktober 2014
# Versie: 1.4

echo -n "Geef een getal van 1 tot 20:"
read vGetal

if [ $vGetal -lt 0 -o $vGetal -gt 20 ]
then
   echo "U gaf geen getal van 1 tot 20"
else
   echo "Het getal dat u gaf is: $vGetal"
fi
                                                              1,1           All
```

if-then-elif

- if-then-elif
  - Je kan if-then-else -structuren in elkaar nesten met if-then-elif
  - Syntax:

```
if
then
elif
then
elif
then
...
else
then
fi
```

Dit is een opbouwende structuur:

Indien expr1
dan ...
anders indien expr2 (en dus niet expr1)
dan ...
anders indien expr3 (en dus niet expr1 en ook niet expr2)
dan ...
anders (wilt dus zeggen in alle andere gevallen)
dan ...
ende indien

if-then-elif - voorbeeld

vwg@laptop: ~/bin

```
#!/bin/bash --

# Auteur: Gert Van Waeyenberg
# Datum:  20 oktober 2014
# Versie: 1.5

echo -n "Geef een getal van 1 tot en met 20:"
read vGetal

if [ $vGetal -le 0 ]
then
   echo "U gaf een te klein getal"
elif  [ $vGetal -lt 10 ]
then
   echo "U gaf een geldig getal in de range van 1 tot en met 9"
elif  [ $vGetal -le 20 ]
then
   echo "U gaf een geldig getal in de range van 10 tot en met 20"
else
   echo "U gaf een te groot getal"
fi
                                                              1,1           All
```

for-loop

```
for <expressie>
do
   commando's
done
```

- for-loop
  - Om de commando's die tussen de do en done van de for-lus staan meerdere keren uit te voeren
  - de expressie bestaat uit een variabele die gebruikt wordt als teller en een range die de telling aangeeft
    - De telling kan bestaan uit
      - losse items (for teller in 1 2 3 4 5)
      - een range (for teller in {1..5}) (for teller in `seq 1 5`)
      - bestanden verkregen door file-globbing (for file in `ls *`)

for-loop - voorbeeld

vwg@laptop: ~/bin

```
#!/bin/bash --

# Auteur: Gert Van Waeyenberg
# Datum:  20 oktober 2014
# Versie: 1.0

echo -n "Tot welk getal wil je dat ik tel:"
read vGetal

echo "OK, hier beginnen we..."

for vTeller in `seq 1 $vGetal`                          # OF {1..$vGetal}
do
   echo -n "$vTeller "
done

echo " "                                              # prompt op volgende lijn
~
                                                                1,1           All
```

vwg@laptop: ~/bin

```
vwg@laptop:~/bin$ ./voorbeeldscript2.sh
Tot welk getal wil je dat ik tel:5
OK, hier beginnen we...
1 2 3 4 5
vwg@laptop:~/bin$
```

for-loop - voorbeeld 2

vwg@laptop: ~/bin

```
#!/bin/bash --

# Auteur: Gert Van Waeyenberg
# Datum:  20 oktober 2014
# Versie: 1.0

echo -n "Van welke directory wil je de bestanden zien?:"
read vDir

echo "Bestanden van $vDir"

for vBestand in `ls $vDir`
do
   if [ -f $vDir/$vBestand ]
   then
      echo -n "$vBestand - "
   else
      echo -en "\e[94m$vBestand\e[39m - "              # directories in blauw
   fi
done

echo " "                                             # prompt op volgende lijn
~
                                                                1,1           All
```

vwg@laptop: ~/bin

```
vwg@laptop:~/bin$ ./voorbeeldscript3.sh
Van welke directory wil je de bestanden zien?:/home/vwg
Bestanden van /home/vwg
bin - Calibre - Library - Desktop - Documents - Downloads - examples.desktop - f
orever.sh - fotowall-background1.jpg - launchy.db - launchy.ini - lijst - lijst2
 - mijnrapport - mijnscript.sh - mijntekst - Music - PDF - persoonlijk - Picture
s - PlayOnLinux's - virtual - drives - Public - selectscript2.sh - selectscript.
sh - Templates - test - test! - testdir - Videos - vmware - weby-icon-cache -
vwg@laptop:~/bin$
```

http://misc.flogisoft.com/bash/tip_colors_and_formatting

while-loop

```
while <voorwaarde>
do
   commando's
done
```

- while-loop
  - Om de commando's die tussen de "do" en "done" van de while staan te blijven herhalen **zolang als** aan de voorwaarde voldaan is
  - als voorwaarde gebruikt men meestal het test-commando
    - Kan bvb. gebruikt worden om de vraag naar invoer te herhalen totdat een juist commando wordt ingegeven

while-loop - voorbeeld

vwg@laptop: ~/bin

```
#!/bin/bash --

# Auteur: Gert Van Waeyenberg
# Datum:  20 oktober 2014
# Versie: 1.0

vGetal=0

while [ $vGetal -lt 1  -o  $vGetal -gt 10 ]
do
   echo -n "Geef een getal van 1 tot en met 10:"
   read vGetal
done

echo "U heeft het getal $vGetal opgegeven!"
                                                    1,1           Top
```

vwg@laptop: ~/bin

```
vwg@laptop:~/bin$ ./voorbeeldscript5.sh
Geef een getal van 1 tot en met 10:-3
Geef een getal van 1 tot en met 10:0
Geef een getal van 1 tot en met 10:11
Geef een getal van 1 tot en met 10:10
U heeft het getal 10 opgegeven!
vwg@laptop:~/bin$
```

while-loop - voorbeeld 2

vwg@laptop: ~/bin

```
#!/bin/bash --

# Auteur: Gert Van Waeyenberg
# Datum:  20 oktober 2014
# Versie: 1.0

vGetal=$(( ( RANDOM % 10 ) + 1 ))
vGok=0

while [ $vGok -ne $vGetal ]
do
   echo -n "Raad een getal van 1 tot en met 10:"
   read vGok
done

echo "U heeft het getal $vGetal geraden !"
                                                    1,1           Top
```

vwg@laptop: ~/bin

```
vwg@laptop:~/bin$ ./voorbeeldscript4.sh
Raad een getal van 1 tot en met 10:1
Raad een getal van 1 tot en met 10:2
U heeft het getal 2 geraden !
vwg@laptop:~/bin$
```

until-loop

```
until <voorwaarde>
do
   commando's
done
```

- until-loop
  - Om de commando's die tussen de "do" en "done" van de until staan te blijven herhalen **totdat** aan de voorwaarde voldaan is
  - als voorwaarde gebruikt men meestal het test-commando
    - Kan bvb. gebruikt worden om de vraag naar invoer te herhalen totdat een juist commando wordt ingegeven

until-loop - voorbeeld

vwg@laptop: ~/bin

```
#!/bin/bash --

# Auteur: Gert Van Waeyenberg
# Datum:  20 oktober 2014
# Versie: 1.0

vGetal=0

until [ $vGetal -ge 1  -a  $vGetal -le 10 ]
do
   echo -n "Geef een getal van 1 tot en met 10:"
   read vGetal
done

echo "U heeft het getal $vGetal opgegeven!"
                                                    1,1           Top
```

vwg@laptop: ~/bin

```
vwg@laptop:~/bin$ ./voorbeeldscript6.sh
Geef een getal van 1 tot en met 10:-1
Geef een getal van 1 tot en met 10:0
Geef een getal van 1 tot en met 10:11
Geef een getal van 1 tot en met 10:10
U heeft het getal 10 opgegeven!
vwg@laptop:~/bin$
```

until-loop - voorbeeld 2

vwg@laptop: ~/bin

```
#!/bin/bash --

# Auteur: Gert Van Waeyenberg
# Datum:  20 oktober 2014
# Versie: 1.0

vGetal=$(( ( RANDOM % 10 ) + 1 ))
vGok=0

until [ $vGok -eq $vGetal ]
do
   echo -n "Raad een getal van 1 tot en met 10:"
   read vGok
done

echo "U heeft het getal $vGetal geraden !"
                                                    1,1           All
```

vwg@laptop: ~/bin

```
vwg@laptop:~/bin$ ./voorbeeldscript7.sh
Raad een getal van 1 tot en met 10:4
Raad een getal van 1 tot en met 10:5
Raad een getal van 1 tot en met 10:6
U heeft het getal 6 geraden !
vwg@laptop:~/bin$
```

script parameters

- script parameters zijn de argumenten die aan een script of een commando worden meegegeven.
  - Bvb: optelsom.sh 15 387 85 97 (parameters 15, 378, 85 en 97)
- Parameters worden opgeslagen in het werkgeheugen. De verwijzing naar de parameter gebeurt via $1, $2, $3,…$9
- Maximaal zijn er 9 verwijzigingen mogelijk.
  - $0 => verwijzing naar de naam van het commando zelf

script parameters

```
student@desktop: ~/bin
#!/bin/bash

# Auteur: Desktop OS
# Datum: 25 oktober 2014
# Versie: 1.0

echo commando-naam: $0
echo parameter 1: $1
echo parameter 2: $2
echo parameter 3: $3
echo parameter 4: $4
echo parameter 5: $5
echo parameter 6: $6
echo parameter 7: $7
echo parameter 8: $8
echo parameter 9: $9
```

```
student@desktop: ~/bin
student@desktop:~/bin$ ./voorbeeldParameters1.sh a b c d e f g h i
commando-naam: ./voorbeeldParameters1.sh
parameter 1: a
parameter 2: b
parameter 3: c
parameter 4: d
parameter 5: e
parameter 6: f
parameter 7: g
parameter 8: h
parameter 9: i
student@desktop:~/bin$
```

script parameters

$#	Verwijst naar het aantal gegeven parameters.
$*	Geeft als resultaat één string waarin alle parameters voorkomen, gescheiden door een delimiter gedefinieerd in de systeemvariabele IFS.
$@	Geeft als output alle parameters waarbij elke parameter als individuele string kan worden gebruikt.
$?	laatste return code
$$	PID van het script

shift through parameters

- slechts 9 parameters ?
- geen melding als de parameter niet bestaat ?
  - $10 wordt aanzien als $1 met en 0 erachter
  - $11 wordt aanzien als $1 met en 1 erachter
  - …

student@desktop: ~/bin

```
#!/bin/bash

# Auteur: Desktop OS
# Datum: 25 oktober 2014
# Versie: 1.0

echo commando-naam: $0
echo parameter 1: $1
echo parameter 2: $2
echo parameter 3: $3
echo "..."
echo parameter 9: $9
echo parameter 10: $10
echo parameter 11: $11
```

student@desktop: ~/bin

```
student@desktop:~/bin$ ./voorbeeldParameters3.sh a b c d e f g h i j k l
commando-naam: ./voorbeeldParameters3.sh
parameter 1: a
parameter 2: b
parameter 3: c
...
parameter 9: i
parameter 10: a0
parameter 11: a1
student@desktop:~/bin$
```

shift through parameters

- shift : de verwijzigingen worden geshift!
  $2 -> $1, $3 -> $2, …

- de waarde van $1 gaat bij iedere shift verloren
  - want $1 krijgt de waarde van $2
  - $2 krijgt de waarde van $3
  - $3 krijgt de waarde van $4
  - …
  - $# (=aantal parameters) wordt ook telkens 1 minder
  - de waarde van $0 (=naam van het commando) blijft behouden

shift through parameters

student@desktop: ~/bin

```
#!/bin/bash

# Auteur: Desktop OS
# Datum: 25 oktober 2014
# Versie: 1.0

echo commando-naam: $0
echo parameter 1: $1
echo parameter 2: $2
echo parameter 3: $3
echo "..."
echo parameter 8: $8
echo parameter 9: $9
echo shift 2x
shift
shift
echo commando-naam: $0
echo parameter 1: $1
echo parameter 2: $2
echo parameter 3: $3
echo "..."
echo parameter 8: $8
echo parameter 9: $9
```

student@desktop: ~/bin

```
student@desktop:~/bin$ ./voorbeeldShift.sh a b c d e f g h i j k l
commando-naam: ./voorbeeldShift.sh
parameter 1: a
parameter 2: b
parameter 3: c
...
parameter 8: h
parameter 9: i
shift 2x
commando-naam: ./voorbeeldShift.sh
parameter 1: c
parameter 2: d
parameter 3: e
...
parameter 8: j
parameter 9: k
student@desktop:~/bin$
```

shift through parameters

student@desktop: ~/bin

```
#!/bin/bash

# Auteur: Desktop OS
# Datum: 25 oktober 2014
# Versie: 1.0

echo commando-naam: $0

vTeller=1
while [ $# -gt 0 ]
do
        echo parameter $vTeller: $1
        let vTeller++
        shift
done
```

student@desktop: ~/bin

```
student@desktop:~/bin$ ./voorbeeldShift2.sh a b c d e f g h i j k l
commando-naam: ./voorbeeldShift2.sh
parameter 1: a
parameter 2: b
parameter 3: c
parameter 4: d
parameter 5: e
parameter 6: f
parameter 7: g
parameter 8: h
parameter 9: i
parameter 10: j
parameter 11: k
parameter 12: l
student@desktop:~/bin$
```

while [$# -gt 0] = zolang als er nog parameters zijn (aftellend naar 0...)

Parameters controleren met regex

vwg@laptop: ~/PXLdemofiles/ScriptingAdvanced

```
#!/bin/bash

if [ $# -eq 0 ]
then
   echo "U gaf geen parameter op. Probeer opnieuw met één parameter..."
elif [[ $1 =~ ^[a-zA-Z]+$ ]]
then
   echo "U gaf een string bestaande uit letters"
elif [[ $1 =~ ^[0-9]+$ ]]
then
   echo "U gaf een getal bestaande uit cijfers"
else
   echo "U gaf een mix van letters, cijfers of andere karakters"
fi
                                                              1,1           All
```

vwg@laptop: ~

```
vwg@laptop:~$ paramscontroleren01.sh
U gaf geen parameter op. Probeer opnieuw met één parameter...
vwg@laptop:~$ paramscontroleren01.sh 123
U gaf een getal bestaande uit cijfers
vwg@laptop:~$ paramscontroleren01.sh abc
U gaf een string bestaande uit letters
vwg@laptop:~$ paramscontroleren01.sh abc123def
U gaf een mix van letters, cijfers of andere karakters
vwg@laptop:~$ paramscontroleren01.sh ù#
U gaf een mix van letters, cijfers of andere karakters
vwg@laptop:~$
```

=~	duidt op een regular expression
[[…]]	nodig als er met regex wordt gewerkt

^[a-zA-Z]+$	^: moet beginnen met [a-zA-Z]: kleine of hoofdletter het vorige 1 of meer keer $: moet eindigen met

Dus: het moet beginnen met een letter, het mogen ook meerdere letters zijn en het moet ook eindigen met een letter

shell functions

- Wat?
  - groep van commando’s
  - wordt aangeroepen door de functienaam

- Waarom?
  - Centraal onderhoud
  - Snelheid
    - functie wordt geprocessed vanuit werkgeheugen en niet vanuit file

shell functions

- een functie kan op 2 manieren gedefinieerd worden:

```
function functienaam
{
    command1
    command2
    command…
}
```

of

```
functienaam
{
    command1
    command2
    command…
}
```

shell functions

- Begin elk script met commentaar (wat doet het script?)
- Plaats ook commentaar doorheen het script om bepaalde stukjes code te verduidelijken (later wil je ook nog gemakkelijk achterhalen waarom iets in het script is opgenomen)
- Gebruik functies om herhaling van stukken code te vermijden

shell functions

- voorbeeld

student@desktop: ~/bin

```
#!/bin/bash

# Auteur: Desktop OS
# Datum: 26 oktober 2014
# Versie: 1.0

function sayHello
{
        echo Hello !
}

echo main script
sayHello
echo end main
```

student@desktop: ~/bin

```
student@desktop:~/bin$ ./voorbeeldFuncties1.sh
main script
Hello !
end main
student@desktop:~/bin$
```

shell functions

- functies werken ook met parameters

student@desktop: ~/bin

```
#!/bin/bash

# Auteur: Desktop OS
# Datum: 26 oktober 2014
# Versie: 1.0

function sayHello
{
        echo Hello $1 !
}

echo main script
sayHello $USER
echo end main
```

student@desktop: ~/bin

```
student@desktop:~/bin$ ./voorbeeldFuncties2.sh
main script
Hello student !
end main
student@desktop:~/bin$
```

shell functions

- de parameters van een functie komen niet overeen met de script parameters

student@desktop: ~/bin

```
#!/bin/bash

# Auteur: Desktop OS
# Datum: 26 oktober 2014
# Versie: 1.0

function slowPrint
{
        echo $1
        sleep $timetowait
        echo $2
        sleep $timetowait
        echo $3
        sleep $timetowait
}

timetowait=$1
slowPrint $4 $3 $2
```

student@desktop: ~/bin

```
student@desktop:~/bin$ ./voorbeeldFuncties3.sh 2 een twee drie
drie
twee
een
student@desktop:~/bin$
```

shell functions

- voorbeeld functie met parameters en gebruik van shift

student@desktop: ~/bin

```
#!/bin/bash

# Auteur: Desktop OS
# Datum: 26 oktober 2014
# Versie: 1.0

function slowPrint
{
        while [ $# -gt 0 ]
        do
                echo $1
                sleep $timetowait
                shift
        done
}

timetowait=$1
shift
slowPrint $@
```

student@desktop: ~/bin

```
student@desktop:~/bin$ ./voorbeeldFuncties4.sh 1 een twee drie
een
twee
drie
student@desktop:~/bin$
```

slowPrint $@, slowPrint $* en slowPrint “$@” geven
een
twee
drie

slowPrint “$*” geeft
een twee drie

shell functions

scope variabelen

student@desktop: ~/bin

```
#!/bin/bash

# Auteur: Desktop OS
# Datum: 26 oktober 2014
# Versie: 1.0

function nieuwevars
{
        x=2
        local y=6
        echo "In de functie is x de waarde $x en y de waarde $y  gegeven"
}

x=1
y=5
echo "bij starten van het script heeft x de waarde $x en y de waarde $y"
nieuwevars
echo "na afloop van de functie heeft x de waarde $x en y de waarde $y"
```

student@desktop: ~/bin

```
student@desktop:~/bin$ ./voorbeeldFuncties5.sh
bij starten van het script heeft x de waarde 1 en y de waarde 5
In de functie is x de waarde 2 en y de waarde 6  gegeven
na afloop van de functie heeft x de waarde 2 en y de waarde 5
student@desktop:~/bin$
```

PXL

exit

- exit-status
  - Elk commando geeft een return code (exit-status) terug aan zijn host-process.
  - Waarde tussen [0..255], 0 = Goed,
  - [1..255] is fout-code.
  - Exit codes with a special meaning http://tldp.org/LDP/abs/html/exitcodes.html
  - Exit status kan bewaard worden door de shell variable ?

```
student@desktop:~$ cd bin
student@desktop:~/bin$ echo $?
0
student@desktop:~/bin$ cd onbestaandedir
bash: cd: onbestaandedir: No such file or directory
student@desktop:~/bin$ echo $?
1
```

exit

- exit
  - beëindigen van een shellscript, exit-status als argument

- return
  - beëindigen van een functie, exit-status als argument

exit

- voorbeeld met exit en return

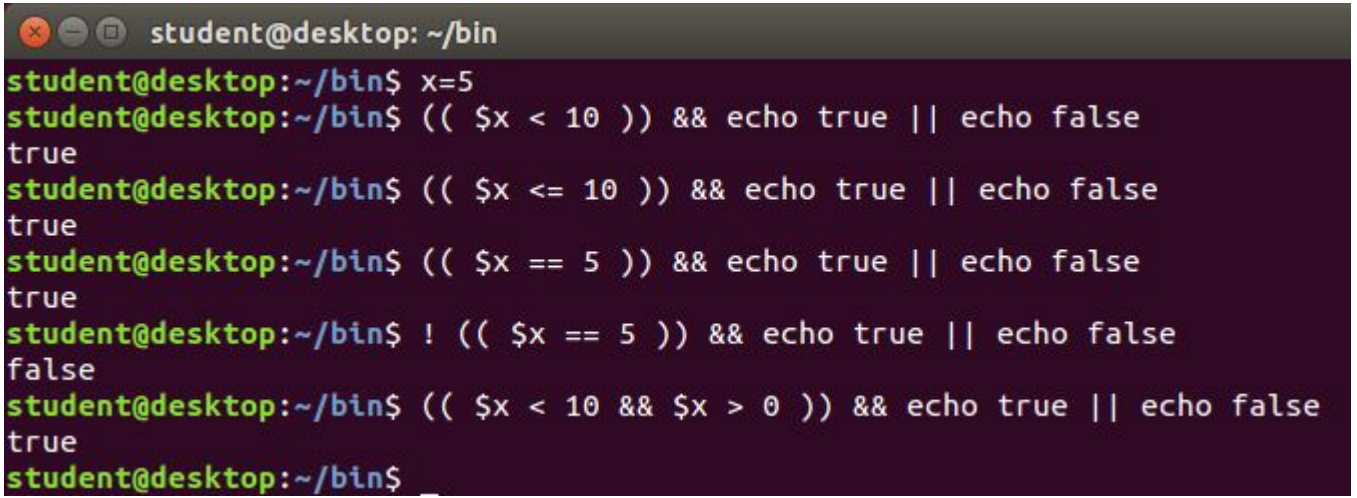
```
student@desktop: ~/bin
#!/bin/bash

# Auteur: Desktop OS
# Datum: 26 oktober 2014
# Versie: 1.0

# functie gebruik van het script
function usage {
        echo "Usage: $0 filename"
        exit 1
}

# functie controleert of file bestaat
function does_file_exist {
        if [ -f $1 ]
        then
                return 0
        else
                return 1
        fi
}

# main, testen van functies
if [ $# -ne 1 ]
then
        usage
fi
```

```
does_file_exist $1
if [ $? -eq 0 ]
then
        echo File found
else
        echo File not found
fi
```

```
student@desktop: ~/bin
student@desktop:~/bin$ ./voorbeeldExit.sh
Usage: ./voorbeeldExit.sh filename
student@desktop:~/bin$ echo $?
1
student@desktop:~/bin$ ./voorbeeldExit.sh test oef1.sh
Usage: ./voorbeeldExit.sh filename
student@desktop:~/bin$ ./voorbeeldExit.sh test
File not found
student@desktop:~/bin$ echo $?
0
student@desktop:~/bin$ ./voorbeeldExit.sh oef1.sh
File found
student@desktop:~/bin$
```

(( ))

- wiskundige tests



- *let ook weer op de spaties: ((␣ <test> ␣))*

student@desktop: ~/bin

```
#!/bin/bash

# Auteur: Desktop OS
# Datum: 25 oktober 2014
# Versie: 1.0

i=1
while (( $# ))
do
        if [[ ! $1 =~ ^[0-9\-][0-9]*$ ]]
        then
                echo "parameter $i: $1 is geen getal"
        elif (( $1 < 10 ))
        then
                echo "parameter $i: $1 is kleiner dan 10"
        else
                echo "parameter $i: $1 is groter of gelijk aan 10"
        fi
        let i++
        shift
done
```

(( ))

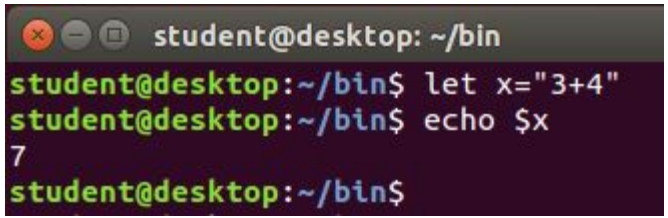
- voorbeeld

student@desktop: ~/bin

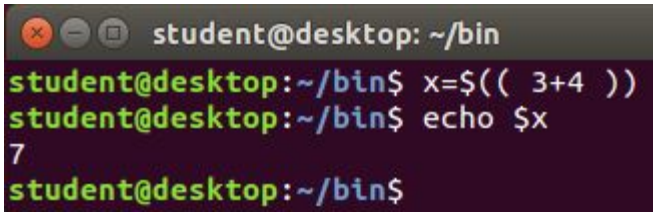
```
student@desktop:~/bin$ ./voorbeeldArithmetic1.sh 0 8 14 a -3 --
parameter 1: 0 is kleiner dan 10
parameter 2: 8 is kleiner dan 10
parameter 3: 14 is groter of gelijk aan 10
parameter 4: a is geen getal
parameter 5: -3 is kleiner dan 10
parameter 6: -- is geen getal
student@desktop:~/bin$
```

let

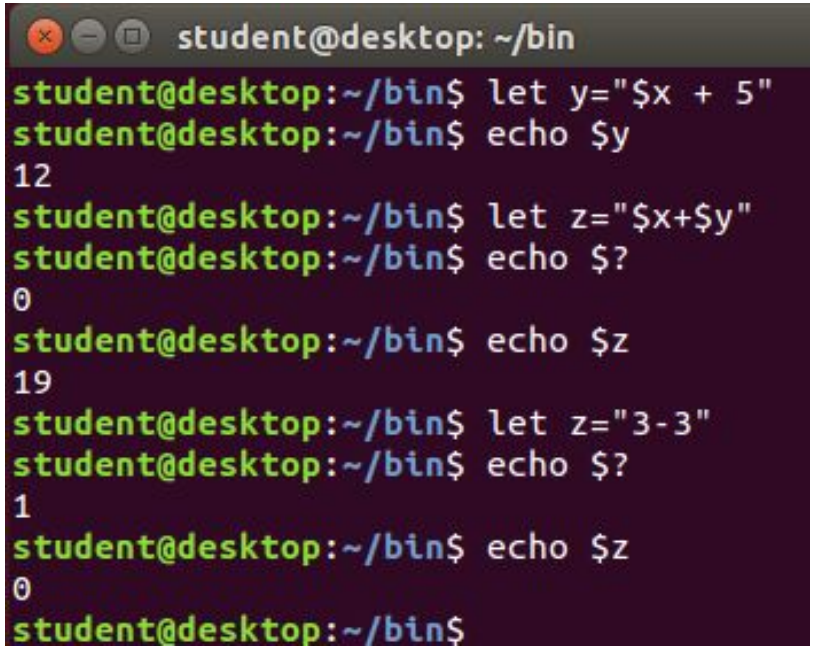
- built-in shell functie
- wiskundige berekeningen



of



let



$? is 0, behalve als de uitkomst 0 is van de wiskundige expressie, in dat geval is $? 1