

# Student-notities

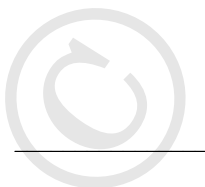
## De shell, sed en awk voor gevorderden

### Voorbeeld-hoofdstuk

03. Control structures



Nijmegen



Copyright © AT Computing 1998, 1999, 2003  
Versie: 3b

**Student-notities**

- De student-notities in dit voorbeeld-hoofdstuk zijn fragmenten uit het dictaat dat bij deze cursus wordt meegeleverd. □



at computing  
The Linux/UNIXperts

## Control structures

Reeds bekend:

```
case string in
  pat1) ... ;;
  pat2) ... ;;
esac

if UNIX-cmd
then
  ...
else
  ...
fi

for var1 in w1 w2 w3 ...
do
  ...
done

while UNIX-cmd
do
  ...
done
```

Figuur 1.

### Student-notities

In plaats van een constructie met een geneste `if`, kun je ook de `elif` gebruiken. Bovendien heeft een goed gebruik van de `;` in plaats van de regelovergang ook nog een positief effect op de leesbaarheid. De drie hierna volgende voorbeelden hebben exact hetzelfde effect:

```
if cmd1
then
    ...
else
    if cmd2
    then
        ...
    else
        ...
    fi
fi
```

```
if cmd1
then
    ...
elif cmd2
then
    ...
else
    ...
fi
```

```
if cmd1; then
    ...
elif cmd2; then
    ...
else
    ...
fi
```

Het voordeel is de grotere leesbaarheid, het kleinere aantal regels — vooral bij lezen achter een terminal heel prettig — en het kleinere aantal inspringingen dat voor de leesbaarheid nodig is.

## Control structures - 2

In plaats van geneste `if` :

```
if UNIX-cmd
then
...
elif UNIX-cmd
then
...
elif UNIX-cmd
then
...
elif UNIX-cmd
then
...
else
...
fi

if UNIX-cmd
then
...
else
if UNIX-cmd
then
...
else
if UNIX-cmd
then
...
else
...
fi
fi
fi
fi
```

Figuur 2.

## Student-notities

Bij zowel de `if` als de `while` wordt de voortgang van het script bepaald door de exitcode van een UNIX-commando. Daarbij geldt dat een exitcode 0 opgevat wordt als "waar" en een waarde ongelijk aan 0 als "onwaar".

Bij beide statements mogen ook meerdere commando's of pipelines staan; we spreken dan van een "list". Commando's en sub-lists in een list worden gescheiden door operatoren die aangeven op welke wijze de combinatie moet worden afgehandeld:

<code>cmd1 ; cmd2</code>	Eerst wordt <code>cmd1</code> uitgevoerd, daarna <code>cmd2</code> . Een return tussen <code>cmd1</code> en <code>cmd2</code> heeft dezelfde betekenis.
<code>cmd1 &amp; cmd2</code>	Commando <code>cmd1</code> wordt in de achtergrond gestart, <code>cmd2</code> wordt gestart.
<code>cmd1 &amp;&amp; cmd2</code>	Betekent <code>cmd1</code> én <code>cmd2</code> : <code>cmd2</code> wordt alleen uitgevoerd als de exitcode van <code>cmd1</code> gelijk aan 0 is.
<code>cmd1    cmd2</code>	Betekent <code>cmd1</code> óf <code>cmd2</code> : <code>cmd2</code> wordt alleen uitgevoerd als de exitcode van <code>cmd1</code> ongelijk aan 0 is.

Een list met meerdere operatoren wordt van links naar rechts afgehandeld. De operatoren `&&` en `||` hebben een hogere prioriteit dan `;` en `&`

De exitcode van een list is de exitcode van het laatst uitgevoerde commando van de list. Bij de background-operator `&` ligt dat iets anders: daar is de exitcode van het op de voorgrond draaiende commando (hier `cmd2`) de exitcode van de hele constructie. De exitcode van het background-proces speelt hier geheel geen rol.

In onderstaand voorbeeld controleren we of `var` de naam van een directory bevat. Alleen dán doen we een `cd` naar deze directory. Controle en `cd` doen we in één keer achter de `if`:

```
if test -d "${var}" && cd "${var}"
then
    pwd
    ls
else
    print "${var} is geen directory, of niet begaanbaar"
fi
```



## Control structures - 3

- waar een commandoregel staat, dus ook achter `if` en `while`, mag een *list* staan:

list: één of meer commando's of pipelines

`;` het eerste commando, dan het tweede  
`cmd1; cmd2`

`&` eerste commando in de achtergrond  
`cmd1& cmd2`

`&&` AND  
`cmd1 && cmd2`  
`cmd2` alleen uitvoeren als exit code van `cmd1` == 0

`||` OR  
`cmd1 || cmd2`  
`cmd2` alleen uitvoeren als exit code van `cmd1` != 0

Exit code van de list is exit code van het laatst uitgevoerde commando uit de list

**Figuur 3.**

### Student-notities

In de shells van het type ksh en bash is er een speciaal statement waarmee je een menu kunt tonen en het antwoord van de gebruiker kunt afhandelen. De z shell kent een soortgelijke constructie, maar die werkt net iets anders (hier niet behandeld).

Een menu wordt op het scherm gezet als een rij teksten (die je zelf bepaalt) met een nummer vóór iedere tekst. Onder het menu komt een vraag te staan.

De te stellen vraag zet je in de shell-variabele PS3, de keuzeteksten geef je op met het select statement. Dat gaat als volgt:

```
PS3="de te stellen vraag"
select varnaam in k1 k2 k3 k4
do
    ...
done
```

Op het scherm verschijnt nu:

- 1) k1
- 2) k2
- 3) k3
- 4) k4

*de te stellen vraag*

Als de gebruiker een antwoord heeft gegeven, worden de statements tussen do en done uitgevoerd. Het menu wordt na ieder antwoord van de gebruiker opnieuw getoond, totdat je met een break uit het select statement springt.

In de variabele REPLY komt het antwoord te staan, letterlijk zoals de gebruiker het heeft ingetikt. Is dat antwoord gelijk aan één van de getoonde nummers, dan wordt bovendien de bijbehorende keuzetekst in de variabele varnaam gezet. Is de keuze niet één van de getoonde nummers, dan wordt varnaam leeggemaakt(!).



## Control structures - 4

- Menu presenteren (alleen ksh en bash):

```
PS3='de te stellen vraag'  
select var in k1 k2 k3 ...  
do  
    ...  
done
```

→

```
1)k1  
2)k2  
3)k3  
...  
de te stellen vraag
```

- het ingetypte antwoord komt in **REPLY**
- als **\$REPLY** een geldig nummer is, dan wordt de bijbehorende tekst in **var** gezet
- als antwoord **<return>** dan menu opnieuw
- uit menu springen met **break**

**Figuur 4.**

### Student-notities

Als voorbeeld laten we hieronder een menu zien waar de gebruiker zowel met nummers als met de keuzetekst kan antwoorden. We tonen altijd de gekozen tekst en zorgen ervoor dat bij het verlaten van het menu de variabele altijd gevuld is met de gekozen tekst.

```
PS3="geef uw keuze"
select var in K1 K2 K3 K4
do
  case "${REPLY}" in
    [1-4]) print "${var}"      nummer gekozen, var is dus gevuld
           break              uit select springen
           ;;
    K[1-4]) print "${REPLY}"
             var="${REPLY}"    geen nummer gekozen, var dus zelf vullen
           break              uit select springen
           ;;
    *)      print foute keuze, opnieuw!
           ;;                  geen break, menu opnieuw tonen
  esac
done
```

Een ander voorbeeld vind je op de sheet.

## Control structures - 5

Voorbeeld van een menu met `select`  
(file `~/selectdemo`):

```
PS3='Your choice please: '  
select answer in plain sorted \  
                'reverse sorted'  
do  
  case ${answer} in  
    plain)      who  
                break  
                ;;  
    sorted)     who | sort  
                break  
                ;;  
    reverse\ sorted) who | sort -r  
                break  
                ;;  
    *)          print 'Invalid choice'  
                ;;  
  esac  
done
```

Figuur 5.

## Student-notities

Een control structure vormt in zijn geheel één statement. Je mag er bijna alles mee doen wat je met ieder statement mag, zoals:

- Gebruiken in een pipeline:

```
who | while read line
do
    verwerk "${line}"
done
```

Deze constructie stelt je dus in staat om meerregelige uitvoer van een commando (bijvoorbeeld een shell-script) regel-voor-regel te laten verwerken.

- De input en/of output redirecten:

```
while read line
do
    verwerk "${line}"
done <infile >uitfile
```

- In de achtergrond starten:

```
for TERM in $(who|awk '{print $2}')
do
    ls -l "/dev/${TERM}"
done >lijst&
```

Je kunt een control structure *niet* als commando gebruiken op een plek waar het een argument van een ander commando is, bijvoorbeeld achter nice of nohup, of na het argument -exec of -ok bij het commando find .



## Control structures - 6

- een control structure is in zijn geheel één statement

mag dus achter een pipeline:

```
who|while read line
do
  verwerk "${line}"
done
```

met input en/of output redirection:

```
while read line
do
  verwerk "${line}"
done <infile >uitfile
```

kan in zijn geheel in de background worden gestart:

```
for term in $(who|awk '{print $2}');
do
  ls -l /dev/${term}
done >lijst &
```

kan niet achter nice of nohup

Figuur 6.

