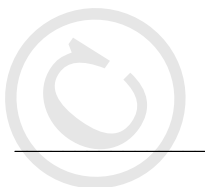


Student-notities
Linux/UNIX deel 2
Voorbeeld-hoofdstuk

02. Shell-programmeren

 **at computing**
The Linux/UNIXperts

Nijmegen



Copyright © AT Computing 2004
Versie: 4d

Student-notities

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

De shell kent een aantal instructies waarmee u commando's kunt herhalen, het uitvoeren van commando's afhankelijk kunt maken van een voorwaarde en waarmee u kunt *springen* in een shell. Gebruikmakend hiervan kunt u echt programmeren: bijvoorbeeld de gebruiker om een stukje input vragen, kijken of hij een geldig antwoord geeft, en daarna een bepaalde reeks commando's uitvoeren. Of: kijken of een commando gelukt is; zo ja, dan de ene vervolgstap doen, en anders een andere.

In de moderne UNIX-systemen wordt veel van shell-files gebruik gemaakt om menusystemen op te zetten. De shell-file controleert wat de gebruiker doet en vertaalt dit dan in gewone UNIX-commando's. Met een beetje handigheid kunt u deze shell files begrijpen, zelf veranderen of opnieuw opzetten.

Programmeren in shell-files

Bij programmeren in shell-files is o.a. nodig:

- Control-structuren

Loop over een lijst woorden:

```
for ... in ... do ... done
```

Selectie:

```
case ... in ... esac
```

Keuze:

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

Conditionele loop:

```
while ... do ... done
```

- Exit code
- Speciale utilities

Conditie testen met **test**

Figuur 1.

Student-notities

Door middel van de `for`-loop kunt u commando's herhalen met telkens een nieuwe waarde van een shell-variabele:

```
for SHELLVAR in 11 22 33 44
do
    echo $SHELLVAR
done
```

zal op het scherm produceren:

```
11
22
33
44
```

Het commando dat tussen `do` en `done` staat, zal vier keer uitgevoerd worden: de eerste keer wordt de shell-variabele `SHELLVAR` gevuld met 11, de tweede keer met 22, dan met 33 en de laatste keer met 44.

In feite is deze `for`-loop het aflopen van de woordenlijst die na het sleutelwoord `in` volgt:

```
for NAAM in woord1 woord2 woord3 .....
do
    .....          voer deze opdrachten herhaaldelijk uit
    .....          met steeds in NAAM de volgende
    .....          waarde uit de lijst van woorden
done
```

Filenaam-expansiesymbolen, commando-substitutie, enzovoort mogen ook weer gebruikt worden:

```
for FILE in *
do
    echo $FILE
done
```

Het symbool `*` zal eerst geëxpandeerd worden tot een complete lijst van filenamen in de huidige directory; daarna zal door middel van de `for`-loop deze woordenlijst afgelopen worden, en elke filenaam wordt door middel van `echo $FILE` apart afgedrukt.

Loop over een lijst woorden: for

```
for var in woord1 woord2 woord3 ...  
do  
    ...  
    ... $var ... } loop body  
done
```

var wordt **woord1**, dan wordt de loop body uitgevoerd.

var wordt dan **woord2**, dan wordt weer de loop body uitgevoerd.

Ook voor **woord3** en verdere woorden uit de lijst.

Voorbeeld 1:

```
for user in anja bert carla diana  
do  
    echo Hallo $user | mail $user  
done
```

Voorbeeld 2 (file: verstuur2):

```
for adres in $(cat $2)  
do  
    mail $adres <$1  
    echo File $1 aan $adres gestuurd  
done
```

Gebruik:

```
$ verstuur2 brief oneven
```

Figuur 2.

Student-notities

Als je shell files bouwt die met behulp van parameters of door een vraag- en antwoordspel met de gebruiker gestuurd worden, zal het vaak voorkomen dat je — al naar gelang de waarde van een parameter of zojuist ingelezen shell-variabele — sprongen in je programma moet maken: bij de ene waarde moet dit gedaan worden, bij een andere wat anders. De case-constructie kan dan handig zijn:

```

case $VARIABELE in
  waarde1)
      .....
      .....
      ;;          betekent: ga naar esac

  waarde2)
      .....
      .....
      ;;          betekent: ga naar esac
      .....
      .....

esac          esac = einde van de case-constructie

```

Bijvoorbeeld: maak een commando `dlist` met twee opties:

- `dlist lang` moet hetzelfde geven als `ls -l`
- `dlist kort` moet hetzelfde geven als `ls`
- `dlist zomaarwat` moet een foutmelding geven

```

case $1 in
  lang)  ls -l
        ;;
  kort)  ls
        ;;
  *)     echo Foute parameter: $1
        ;;
esac

```

Het labeltje `*` betekent: al het andere. Als `$1` niet gelijk is aan (letterlijk) `lang` of `kort` worden de hierna volgende commando's uitgevoerd. Je mag bij de waarden gebruik maken van dezelfde expansiesymbolen als bij de filenaam-generatie (vandaar dat `*` "alles" betekent). Bovendien mag je door middel van `|` verschillende alternatieven scheiden die moeten leiden naar dezelfde reeks commando's. Het commando `dlist` kan nu ook de parameters `long` (naast `lang`) en `short` (naast `kort`) begrijpen (zie voorbeeld op de volgende sheet).

Selectie uit tekstpatronen: case

```

case tekst in
patroon1) cmd1
...
;;
patroon2) cmd2
...
;;
patroon3) cmd3
...
;;
...
esac

```

Als dit past op dat, voer dan dat uit.

Anders, als het past op dit, voer dan dat uit.

Anders, ...

patroon... is een filenaam-patroon (met * ? en [chars] er in).
pat1|pat2 betekent: past op pat1 óf pat2

- Het eerstpassende patroon wint
- Geen enkele past, dan geen enkele uitgevoerd
- Vangnet voor "alle andere gevallen":

```

*)
...
;;

```

Figuur 3.

Student-notities

Een ander voorbeeld: het volgende commando kopieert files naar een directory met de naam /home/ikke/backup, maar zal per file apart om toestemming vragen alvorens dit te doen. Geldige antwoorden zijn: j, J, ja, Ja, JA, n, N, nee, Nee en NEE; bij een fout antwoord geeft het commando een boodschap en kopieert niet:

```
for i in *
do
    echo File $i kopieren?
    read antwoord

    case $antwoord in
    j|J|ja|Ja|JA)    cp $i /home/ikke/backup
                    echo $i gekopieerd
                    ;;
    n|N|nee|Nee|NEE) echo $i niet gekopieerd
                    ;;
    *)              echo Fout antwoord
                    echo $i niet gekopieerd
                    ;;
    esac
done
```

Als je alle antwoorden die beginnen met j, J, n en N wilt accepteren, had je het (gebruikmakend van filenaam-expansiesymbolen) ook zo kunnen schrijven:

```
.....
[jJ]*) cp $i /home/ikke/backup
.....
[nN]*) echo Fout antwoord
.....
```

Als laatste opmerking: de shell loopt de diverse waarden in de volgorde af zoals ze in de file staan; bij de eerste passende stopt hij, voert de daarop volgende commando's uit en zoekt daarna *niet meer* de volgende waarden af (ook al zouden ze eveneens passen):

```
.....
case $antwoord in
*)          echo HIER KOM JE NU ALTIJD TERECHT
            ;;
ietsanders) echo EN HIER DUS NOOIT
            ;;
```

Selectie met case: voorbeeld

Voorbeeld (file: dlist):

```
case $1 in
short|kort)  ls;;
l[ao]ng)    ls -l;;
*)
  echo "Gebruik: $0 kort|lang" >&2
  echo "Usage: $0 short|long" >&2
  ;;
esac
```

Zo is al enige parametercontrole mogelijk
(file: count2):

```
case $# in
0|1)
  echo "Use: $0 text infile..." >&2
  ;;
*)
  txt="$1"
  shift
  grep -c -- "$txt" @$@
  ;;
esac
```

Figuur 4.

Student-notities

Elk commando dat stopt, zal aan de shell een *exit-code* teruggeven. Deze exit-code is een klein geheel getal waarmee het programma aan de shell doorgeeft of het zijn taken al dan niet met succes heeft kunnen uitvoeren. Het is de taak van de programmeur om dat correct in te bouwen.

Een exit-code 0 betekent (bij conventie): alles is goed afgelopen. Een exit-code ongelijk 0 betekent: om een of andere reden kon ik de gestelde opdracht niet goed uitvoeren. Bij een exit-code ongelijk 0 is de precieze waarde niet van belang (de programmeur heeft er waarschijnlijk voor zichzelf een betekenis aan gegeven).

De exit-code krijg je normaliter nooit onder ogen. De exit-code van het laatst uitgevoerde commando komt in shell-variabele \$? te staan en als je die wilt zien, dan zul je daar dus expliciet om moeten vragen. Kijk eens naar het volgende voorbeeld:

```
$ ls /etc/passwd
/etc/passwd
$ echo $?
0
$ ls /etc/passwoord
ls: /etc/passwoord: No such file or directory
$ echo $?
2
$ echo $?
0
```

De eerste keer dat je het `ls`-commando uitvoert, doet `ls` wat je verwacht: je krijgt de gevraagde uitvoer. Wanneer je dan kijkt naar de variabele `$?` staat die op 0; alles is immers goed gegaan.

De tweede keer dat je het `ls`-commando uitvoert, geeft `ls` een foutboodschap: je krijgt de gevraagde uitvoer niet, de file bestaat niet. Wanneer je dan kijkt naar de variabele `$?` staat die op 2. Wanneer je meteen daarna weer het commando `echo $?` uitvoert, krijg je een ander resultaat. Misschien eerst verrassend, maar wel verklaarbaar. De variabele `$?` bevat immers de exit-code van het *laatste* commando. Het laatste commando is in dit geval de vorige `echo`, en die ging goed.

Voor zelfgeschreven shell-files is het ook mogelijk om een exit-code te genereren. De default exit-code die een shell-script afgeeft, is de exit-code van het laatst uitgevoerde commando. Wil je een exit-code genereren om bijvoorbeeld te kunnen aangeven of het script goed of fout is gegaan, dan gebruik je het commando `exit`:

```
case $# in
2) .....
;;
*) echo $0: je moet twee argumenten meegeven
exit 1
;;
esac
```

Exit code

Getal, door een programma aan de oproeper teruggeven.

- Op te vragen via `$?`
- In shell file op te geven met bijv. `exit 3`
- Getal tussen de 0 en 255
- 0 betekent: het is goed gegaan
- Ongelijk 0 betekent: het is fout gegaan
- Als geen exit code opgegeven: exit code van laatst uitgevoerde commando

Probeer:

```
$ ls -l videmo      Die file is er, dus de
$ echo $?          exit code zal 0 zijn

$ ls -l /x/y/z     Die file is er niet, dus de
$ echo $?         exit code is ongelijk 0
$ echo $?         Deze exit code is 0!

$ cmd3            Exit code willen we houden,
$ ec3=$?         dus die slaan we op
... $ec3 ...     om hem later te gebruiken
```

Ieder proces geeft een exit code.

Figuur 5.

Student-notities

We kunnen de exit-code van een programma of een ander shell-script gebruiken. U kunt de exit-code testen en naar gelang van de uitkomst bepaalde volgende commando's al of niet uitvoeren. U hoeft hiervoor de variabele \$? niet te gebruiken. Bijvoorbeeld: verwijder alle files in de huidige directory waarvan de inhoud het woordje UNIX bevat:

```
for i in *
do
    if grep UNIX $i > /dev/null
    then
        rm $i
    fi
done
```

Door middel van de `for`-loop worden alle files in de huidige directory afgelopen. Voor elke file wordt met behulp van het commando `grep` gekeken of het woordje UNIX hierin voorkomt; de normale uitvoer van `grep` schrijven we naar de speciale file `/dev/null` (de prullenbak van UNIX) zodat we deze uitvoer niet op het scherm krijgen. De exit-code van `grep` zal 0 zijn als hij iets vindt, en ongelijk 0 als hij niets in de file vindt. Alleen als `grep` een exit-code gelijk 0 geeft, wordt het commando `rm $i` uitgevoerd: hiervoor zorgt de `if-then-fi` constructie (de `else` is optioneel):

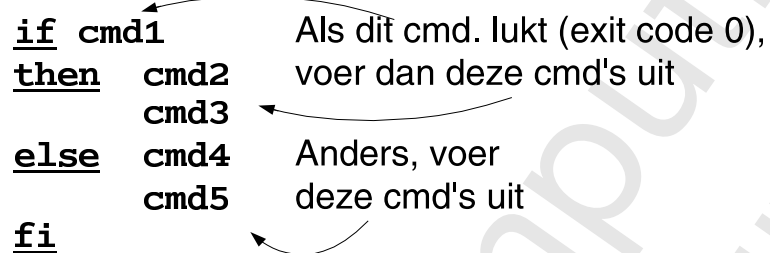
```
if gewoon-UNIX-commando
then
    ..... dit wordt uitgevoerd wanneer de
    ..... exit-code gelijk is aan nul, d.w.z.
    ..... wanneer het commando goed ging
else
    ..... dit wordt uitgevoerd wanneer de
    ..... exit-code ongelijk is aan nul, d.w.z.
    ..... wanneer het commando fout ging
fi
```

Dus met `else`-constructie:

```
if grep UNIX $i > /dev/null
then
    echo $i bevat het woord UNIX
    rm $i
else
    echo $i bevat NIET het woord UNIX
fi
```

Tweewegkeuze: if - then - else - fi

```
if cmd1      Als dit cmd. lukt (exit code 0),  
then cmd2    voer dan deze cmd's uit  
          cmd3  
else cmd4    Anders, voer  
          cmd5    deze cmd's uit  
fi
```



else-gedeelte mag weggelaten worden.

Voorbeeld (file: testuser):

```
if grep -- $1 /etc/passwd >/dev/null  
then  
    echo User $1 known  
else  
    echo User $1 unknown  
fi
```

Voorbeeld (file: safecopy):

```
if cp $1 $HOME/savedir/$1  
then  
    echo Safecopy of $1 made >&2  
else  
    echo No safecopy of $1 made! >&2  
fi
```

Figuur 6.

Student-notities

In shell-files zul je in het bijzonder gebruik maken van de exit-code van het speciale commando `test`. In feite *doet* dit commando niets, het *test* alleen of datgene wat je als parameters meegeeft *waar* is of *onwaar*. Met andere woorden: door aan `test` een reeks argumenten mee te geven stel je in feite een vraag. En `test` beoordeelt die vraag op waar/onwaar. Door middel van exit-code 0 rapporteert `test` aan de shell dat het waar is, door middel van exit-code 1 dat het onwaar is.

Bijvoorbeeld: gooi alle files onder de huidige directory weg die een lengte nul hebben:

```
for i in *
do
    if test -s $i
    then
        echo file $i heeft lengte ongelijk nul
    else
        echo file $i heeft lengte nul
        rm $i
    fi
done
```

De parameter `-s` van `test` betekent: ga na of het volgende argument de naam is van een gewone file met een lengte ongelijk 0; als dit het geval is, geef exit-code waar (gelijk 0) terug, anders exit-code onwaar (ongelijk 0) terug.

Met `test` kunt u testen op alle mogelijke zaken. Een voorbeeld van de vragen die je kunt stellen ten aanzien van file-objecten:

- `test -s filenaam` Bestaat de file en heeft ze lengte > 0 ?
- `test -r filenaam` Heeft de file voor mij leespermissie?
- `test -w filenaam` Heeft de file voor mij schrijfpermissie?
- `test -d filenaam` Bestaat de file en is het een directory?
- `test -f filenaam` Bestaat de file en is het een regular (gewone) file?

Er is een alternatieve notatie voor `test` die veel wordt gebruikt. In plaats van:

```
test expressie
```

mag je ook schrijven:

```
[ expressie ]
```

waarbij rondom elk van de blokhaken spaties *moeten* staan. Het sleutelwoord `test` vervalt hierbij. Deze notatie komt meer overeen met wat in andere programmeertalen de gewoonte is.

Conditie testen - test

Achter `if` moet een commando staan.

Maar ik wil:

- Als dit een file is, dan ...
- Als er minder dan twee parameters zijn, dan ...

Daarvoor: commando `test` geeft exit code terug:

geeft exit code 0 als
`data3` een file is

```
if test -f data3
then echo data3 is een file
fi
```

Allerlei tests mogelijk, bijvoorbeeld:

- testen op soort object (file, directory, ...)
- testen op lees- of schrijf-recht
- getallen vergelijken
- teksten vergelijken (alleen gelijk/ongelijk)
- combinaties van tests ("en", "of", "niet")

Modernere notatie tussen `[` en `]` :

```
if [ -f data3 ]
then echo data3 is een file
fi
```

whitespace vereist!

Figuur 7.

Student-notities

In het resterende deel van dit hoofdstuk worden de details uitgelegd met betrekking tot het commando `test` en komt de controle-structuur `while` uitgebreid aan de orde.



at computing
The Linux/UNIXperts