COMPUTERSYSTEMEN 2
Linux

Inhoudsopgave

1 BASH INLEIDING

"Più facilmente si contasta al principio che alla fine"
-- Leonardo da Vinci

1.1 Doelstelling van de cursus

Deze cursus is een vervolg op de cursus Linux uit het vak Computersystemen 1.
Een basiskennis van Linux is dus vereist.

De cursus wordt als "begeleide zelfstudie" aangeboden. Je moet dus vooral **zelf** de oefeningen kunnen oplossen. De oplossingen worden niet ter beschikking gesteld. Denk ook niet te snel dat je alles kent omdat je een oplossing gekregen hebt. Je kan altijd wel bij de docent terecht bij problemen met een bepaalde oefening.

1.2 Opmaak en benodigdheden

a) Opmaak van de cursus

Bij de voorbeelden wordt de standaard prompt van de systemen getoond.
Dit geeft voor Linux systemen:

- een dollarteken **$** voor gewone gebruikers
- een hekje **#** voor de root gebruiker

Dit moet je dus **niet** ingeven om het commando uit te voeren.

Bij de vorm van commando's worden optionele delen tussen vierkante haken geplaatst: bv. **`ls [bestandsnaam]`** (Dit commando geeft een directory listing)
Toetsen of toetscombinaties worden tussen **< >** weergegeven.
bv **<CTRL-C>** betekent dat je de **CTRL** toets én de toets **C** moet indrukken.
Drie puntjes in een schermoutput **...** geven aan dat er wat tekst is weggelaten.

b) Benodigdheden voor de cursus

- Je installeert het Linux systeem bij voorkeur op je harde schijf (best na de nodige backups van je bestaande partities en bestanden).
- Als het installeren op je harde schijf problemen geeft met met je hardware kan je ook Linux installeren in Virtualbox/VMware. Zorg dan zeker voor een backup van je virtuele harde schijf!
- Je kan ook Linux installeren door de installatie in Windows te starten (wubi.exe). Linux draait dan vanuit een bestand op de C schijf. In de praktijk bleek dit niet de meest veilige optie voor je Linux systeem. Als het bestand corrupt wordt (bv na een Linux OF Windows crash) dan ben je alles kwijt. Hetzelfde geldt trouwens voor een virtueel systeem. Als de bestanden van je virtuele schijf corrupt zijn, dan ben je alles kwijt.

- Je kan ubuntu/kdguntu ook gewoon opstarten met een LiveCD (zonder te installeren). Let er dan wel op dat je je bestanden bewaart op een bestaande schijf (bv de NTFS partitie van Windows of een USB stick)

  Je mag uitgaan van de standaard Ubuntu 12.04 desktop editie (64 bit) en daarin de extra pakketten installeren.

Wat nodig is voor het examen is dat je op het draadloos netwerk kunt. Als allerlaatste optie kan je dus ook Linux installeren in Virtualbox/VMWare onder Windows/Mac.

1.3 Werken op Ubuntu Linux

1.3.1 Inloggen op Ubuntu

Verbinden met je Linux server kan op volgende manieren:

- Bij het booten van de cd logt Ubuntu automatisch in met de gewone gebruiker ubuntu.
- Als je te weinig rechten hebt om iets te bekijken, uit te voeren of aan te passen, dan helpt het meestal om root gebruiker te worden. Dit kan je worden door in een shell het commando **sudo su** te geven. Wanneer je gedaan hebt als root gebruiker tik je gewoon exit en je bent terug gewone kdguntu gebruiker.
  Je MOET root gebruiker zijn (of toelating hebben via het sudo commando) om services te starten en te stoppen, configuraties aan te passen of pakketten te installeren, kortom voor alles wat je niet door je bomma wil laten uitvoeren.

```
kdguntu@kdguntu:~$ whoami
kdguntu
kdguntu@kdguntu:~$ sudo su
root@kdguntu:/home/kdguntu# whoami
root
root@kdguntu:/home/kdguntu# exit
kdguntu@kdguntu:~$
```

In het kort kan je `sudo` gebruiken om een bepaald commando uit te voeren als root gebruiker. De commando's die je zo kan uitvoeren zijn hiervoor gemerkt met een speciale bit. Verder staat er welke gebruiker welke commando's mag gebruiken in het bestand /etc/sudoers.

Wanneer ik de file-manager nautilus als root wil gebruiken kan dat als volgt vanuit een terminal (met gksudo, de grafische variant van sudo) :

```
kdguntu@kdguntu:~$ gksudo nautilus &
[sudo] password for kdguntu:
Initializing gnome-mount extension
```

Hiervoor moet je wel voor de standaard kdguntu gebruiker een paswoord ingesteld hebben. Dit kan met het commando `passwd kdguntu`.

Een alternatief voor het opstarten van nautilus als root kan zijn dat je eerst met `sudo su` root gebruiker wordt en daarna als root gebruiker nautilus opstart:

```
kdguntu@kdguntu:~$ sudo su
[sudo] password for kdguntu:
root@kdguntu:/home/kdguntu# nautilus
```

Je ziet dat je voor het commando nautilus dan geen sudo meer hoeft te zetten. Je mag dit wel, maar het moet niet.
Je kan opnieuw je terminal niet meer gebruiken. Wil je toch je terminal opnieuw gebruiken dan voeg je een ampersand toe aan je commando. Het proces nautilus wordt dan als apart proces (eigenlijk in de achtergrond) opgestart.

```
kdguntu@kdguntu:~$ sudo su
[sudo] password for kdguntu:
root@kdguntu:/home/kdguntu# nautilus &
```

1.4 Installeren van software

Update van de pakketlijst:

```
root@kdguntu:~# sudo apt-get update
```

Zoeken naar het pakket vim:

```
root@kdguntu:~# apt-cache search vim
...
tmux - terminal multiplexer
vim - Vi IMproved - enhanced vi editor
vim-common - Vi IMproved - Common files
...
```

Installeren van vim:

```
root@kdguntu:~# sudo apt-get install vim
```

Soms zijn er problemen met de mirror be.archive.ubuntu.com. Deze kan je als root vervangen in /etc/apt/sources.list door archive.ubuntu.com en daarna terug updaten met sudo apt-get update.
Of grafisch in Ubuntu Software Center in de menu Edit -> Software Sources.
Kies daar voor Download from: “Main server” in plaats van “Server for Belgium”

1.5 Schrijven van shell scripts

1.5.1 De bash shell

De shell is de tekstmodus interface naar je besturingssysteem. Standaard bevat deze een aantal commando's zoals cd, pwd, ls, echo,...

Handige bash (Bourne-Again SHell) eigenschappen zijn:

- Bijhouden van de **history** (geschiedenis)
  Vorige commando's krijg je door je pijltje naar boven te doen.
- **Command Completion**. In plaats van de volledige bestandsnaam in te tikken, geef je de eerste letters en druk je <TAB>. De shell vult automatisch de bestandsnaam aan tot waar mogelijk (soms doe je <TAB><TAB>
- De ondersteuning van "kleuren"

1.5.2 De terminal

Standaard is je tekstmodus "command prompt" (CMD.EXE in Windows) gnome-terminal. Een andere mogelijke prompt is xterm en terminator
Binnen en tussen grafische programma's kan je copiëren met <CTRL><C> en plakken met <CTRL><V>. In de gnome-terminal moet dit met <CTRL><SHIFT><C> en <CTRL><SHIFT><V> respectievelijk.

1.5.3 De vi editor

Om een file te bewerken of om een nieuwe file aan te maken tik je: `$ vi filename`

Vi heeft 2 modes:
De commando mode: voor bewaren / zoeken /verwijderen
De tekst mode: voor intikken van tekst (insert) of aanvullen (append)

a) vi command mode

Om er absoluut zeker van te zijn dat je in de commando mode zit, tik je 2x ESC voor een commando.
De meest gebruikte commando's:

TOETS	*FUNCTIE*
:w	write file
:wq	write en quit file
:q!	quit zonder bewaren
/zoekstring	een "find" Door n te tikken krijg je de volgende (next) string in de tekst
x (op een char)	verwijdert het karakter
dd (voor een lijn)	CUT van de lijn
yy (voor een lijn)	COPY van de lijn
p (voor een lijn)	PASTE van de lijn

b) vi text mode

Om in tekst mode te geraken typ je de ‘i’ van insert ergens in de tekst. Vanaf daar kan je tekst intikken en verwijderen met <BACKSPACE>. Om eerder geschreven tekst te verwijderen moet je terug in commando mode.
In plaats van 'i' kan je ook de 'a' van append kiezen om karakters toe te voegen (bv aan het einde van de lijn)
vi heeft betere versies (bv vim) die ook de delete toets en de backspace toets verstaan, waardoor je niet meer uit de tekstmode hoeft te gaan

c) vi tips

a) Ik kan niets meer doen in VI
   druk ESC en :q!
b) Ik kan niet met de pijltjes bewegen in VI
   Je terminal werkt niet met VT100 arrows, doe "apt-get install vim"
c) Ik wil iets bewaren, maar :wq lukt niet
   Je bestand staat waarschijnlijk in read-only-mode. Als root kan je :wq! doen om toch een verandering aan een read-only bestand te bewaren. Als gewone gebruiker kan je enkel met :q! (zonder te bewaren) de vi editor verlaten.
d) In het bestand ~/.vimrc zijn volgende lijnen handig:
   set nocompatible # ook werken met pijltjes
   syntax enable # syntax highlighting aan

1.5.4 Uitvoeren van een shell script

Je shell kan shell-scripts uitvoeren. Een script is een uitvoerbaar tekstbestand met volgende voorwaarde opdat je het kan uitvoeren:

1. Het bestand moet executable zijn
   In een terminal kan dit met volgend commando:
   `chmod +x scriptfile.sh`
2. De eerste regel van het bestand moet de volgende zijn:
   `#!/bin/bash`
3. Opstarten door in een terminal volgend commando te geven
   `./scriptfile.sh`

Opmerkingen:

- Je merkt dat je niet gewoon `scriptfile.sh` kunt ingeven om het script te starten. Dit komt omdat in het pad (te zien met `echo $PATH`) geen punt staat (punt is je huidige directory). Je huidige directory opnemen in je PATH is geen goed idee aangezien een trojaans paard anders in je home directory bv het virusbestand met de naam "ls" schrijft, dat dan voorrang krijgt op de echte /bin/ls.

1.6 Speciale variabelen

$0
Geeft de bestandsnaam van het bash script

```
#!/bin/bash
# Voorbeeld tonen scriptnaam
echo "Dit script heet $0 "
```

Mooier (zonder directorynaam) gebruik je `` `basename` `` :

```
echo "Dit script heet `basename $0`"
```

$1
Toont het eerste argument 1. De argumenten geef je in op de command line

$#
Toont het aantal argumenten dat is meegegeven op de command line

$?
Toont de exit code van het laatst uitgevoerde commando. Exit code 0 is gelukt. Niet nul is mislukt

$@
Toont ALLE argumenten die zijn meegegeven op de command line. Alle parameters zijn een aparte string.

```
 #!/bin/bash
# Functie:      Gebruik van stdin of file input met cat $@
# Argumenten:   Argument $1 MAG meegegeven worden en is een bestand
#                  Script mag ook als standaard input tekst krijgen

IFS=$'\n'
for lijn in $(cat "$@")
  do
    echo $lijn
 done
unset $IFS
```

```
Output:

kdg@kdguntu$ echo -e "lijn 1 test\nlijn 2 test" | ./dollaradd.sh
lijn 1 test
lijn 2 test
kdg@kdguntu$ ./dollaradd.sh /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
...
```

$*
Toont ALLE argumenten die zijn meegegeven op de commandline als 1 grote string

$$
Bevat de proces ID van het huidige bash proces. Vaak gebruikt als unieke temp bestandsnaam.

$_
Bevat het laatst gebruikte argument

$!
Bevat de proces ID van het laatste achtergrondproces

```
#!/bin/bash
# Functie: tonen gebruik interne variabelen
# Naam:    moeder.sh

echo Dit moeder script heet $0
echo Argument 1 is $1
# eindeloos child proces schrijven en opstarten
echo '#!/bin/bash' > child.sh
echo 'while true; do echo "kind $$"; sleep 1; done' >> child.sh
echo Laatste argument: $_   # Dit is van echo de lijn hiervoor
chmod +x child.sh
./child.sh &                  # Opstarten in achtergrond
sleep 2
echo kind nr $! wordt door moeder afgemaakt
kill $!                       # PID laatste achtergrondproces (child.sh)
echo moeder $0 nr $$ pleegt zelfmoord
kill $$                       # PID eigen proces (moeder.sh)
```

```
Output:
Dit script heet ./moeder.sh
Argument 1 is test
Laatste argument: while true; do echo "kind $$"; sleep 1; done
kind 2400
kind 2400
kind nr 2400 wordt door moeder afgemaakt
moeder ./moeder.sh nr 2398 pleegt zelfmoord
Terminated
```

1.7 Speciale karakters

Om interpretatie van speciale karakters te vermijden, gebruik je backslashes of single quotes..

```
echo "De variabele \$HOME heeft als waarde $HOME"
echo 'Hier wordt alles letterlijk weergegeven, ook $HOME"
```

Dit geeft:

```
De variabele $HOME heeft als waarde /home/kdguntu
Hier wordt alles letterlijk weergegeven, ook $HOME
```

1.8 File I/O

1.8.1 Inlezen van een bestand

```
OLD_IFS="$IFS"
IFS=$'\n'
 for lijn in `cat /etc/passwd` do
  echo $lijn
done
IFS="$OLD_IFS"
```

1.8.2 Schrijven naar een bestand

```
jancelis@kdguntu:~$ echo "Dit maakt een nieuw bestand" > een_bestand.txt
jancelis@kdguntu:~$ echo "Dit voegt lijnen toe" >> een_bestand.txt
jancelis@kdguntu:~$ uname -s >> een_bestand.txt
jancelis@kdguntu:~$ cat een_bestand.txt
Dit maakt een nieuw bestand
Dit voegt lijnen toe
Linux
```

1.9 Opstarten externe programma's

Op volgende manier wordt de output van een programma weggeschreven in een variabele.

```
# Opgelet met backquotes (ALTGR-mu)
 mijnvar=`uname -s`
 echo $mijnvar
```

1.10 Operators

a) Wiskundige operators

Operator	Voorbeeld	Resultaat	Definitie
+	echo $((7+5))	12	Optellen
-	echo $((7-5))	2	Aftrekken
*	echo $((7*5))	35	Vermenigvuldigen
/	echo $((7/5))	1	Delen
**	echo $((7**5))	16807	Exponent
%	echo $((7%5))	2	Modulus

b) Relationele operators strings

Operator	Voorbeeld
=	"Hallo" = "Hallo"
!=	"Jan" != "Eddy"

Opmerking:
Case insensitive vergelijking van strings doe je door de shelloptie nocasematch aan te schakelen

```
shopt -s nocasematch  #aanschakelen

if [ "jan.celis" = "Jan.Celis" ]; then echo "Match!" ; fi

shopt -u nocasematch  #uitschakelen
```

Operator	Voorbeeld
-eq	5 -eq 5
-ne,	7 -ne 2
-lt	4 -lt 7
-gt	7 -gt 4
-le	7 -le 7 && 7 -le 1
-ge	11 -ge 11 && 12 -ge 11

c) Logische operators

&&	of	-a	Vergelijkingen tussen strings combineren met logische AND
||	of	-o	Vergelijkingen tussen strings combineren met logische OR
!			Negatie

1.11 Debug in bash

Debuggen door een trace te starten van een shell script, kan door het script op te roepen met de optie -x

```
jancelis@kdguntu$ bash -x ./debug.sh
```

of door in de eerste regel van het script de optie -x toe te voegen

```
#!/bin/bash -x
var=1
echo $(var+2)
```

```
Output:
+ var=1
+ echo 3
3
```

2 HERHALING LOOPS

while true;
do echo "Linux can do infinite loops in 5 seconds';
echo '-- Linus Torvalds';
done

2.1 test

Doel: Testen met condities

Vorm: `test conditie`

Uitleg: De uitdrukking wordt geevalueerd en indien waar wordt nul als waarde teruggegeven. De volgende operanden mogen gebruikt worden in de uitdrukking expr samen met -a (and), -o (or), ! (not)

Test is de basis van een if structuur. Het commando test wordt meestal weggelaten. Alle opties en uitleg vind je met `man test`

Opties:

Opties test	Uitleg
`-r [file]`	Leesbaar bestand
`-w [file]`	Schrijfbaar bestand
`-x [file]`	Uitvoerbaar bestand
`-f [file]`	Normaal bestand
`-d [file]`	Directory
`-s [file]`	Bestand met grootte > 0 (dus niet leeg)
`-z [string]`	Lengte van de string is nul (dus leeg)
`-n [string]`	Lengte van de string is niet nul (dus niet leeg)
`s1 = s2`	String s1 is gelijk aan string s2
`s1 != s2`	String s1 is NIET gelijk aan string s2
`n1 -eq n2`	De gehele getallen n1 en n2 zijn gelijk in waarde (**eq**ual)
`n1 -ne n2`	De gehele getallen n1 en n2 zijn verschillend (**n**ot **e**qual)
`n1 -gt n2`	n1 is groter dan n2 (**g**reater **t**han)
`n1 -lt n2`	n1 is kleiner dan n2 (**l**ess **t**han)
`n1 -ge n2`	n1 is groter dan of gelijk aan n2 (**g**reater /**e**qual)
`n1 -le n2`	n1 is kleiner dan of gelijk aan n2 (**l**ess /**e**qual)

Voorbeeld:

```
kdguntu@kdguntu:~$ test -r .bash_profile
kdguntu@kdguntu:~$ test -r /etc/passwd
```

2.2 if

Standaard gebruikt if het test commando als volgt:

```
#!/bin/bash
# Functie: Als /etc/passwd bestaat de inhoud tonen

if test -f "/etc/passwd"
  then
    cat /etc/passwd
fi
```

De functie test wordt door de echte pro's (zoals wij) vervangen door vierkante haakjes. Vandaar dat bovenstaand programma er normaal zo uit ziet:

```
#!/bin/bash
# Functie: Als /etc/passwd bestaat de inhoud tonen

if [ -f "/etc/passwd" ]
  then
    cat /etc/passwd
fi
```

Je zal merken dat shell scripts zeer kritisch zijn in verband met spaties, wanneer je lussen gebruikt. Geheugensteuntje bij bash is ALTIJD een spatie voor en na de vierkante haken.

```
#!/bin/bash
# Functie: Als configuratiebestand bestaat een lijn toevoegen,
#          anders een nieuw bestand aanmaken

conffile="myprogram.conf"
if [ -f "$conffile" ]
  then
    echo "# Nieuw bestand aanmaken" > "$conffile"
  else
    echo "# toevoegen" > "$conffile"
fi
cat "$conffile"
```

2.3 while

while met een teller

```
 #!/bin/sh
 teller=0
 while [ $teller -lt 10 ]
  do
  echo -n "$teller "
  let teller+=1      # eentje bijtellen
  done
 echo -e "\nEinde"
```

Een while kan je gebruiken om lijn per lijn gegevens uit een bestand te lezen:

```
#!/bin/sh
while read lijn in `cat gebruikersnamen.txt`
do
  echo "mkdir /home/$lijn"
done
```

2.4 case

Twee keer ;; gebruik je om uit de case-lus te stappen
De string (of de variabele) wordt opeenvolgend vergeleken met elk patroon en bij overeenkomst wordt de commando-lijst uitgevoerd.

```
  echo -n "Geef de naam van het beest: "
  read BEEST
  echo -n "De $BEEST heeft "
  case $BEEST in
     paard | kat | koe) echo -n "vier";;
     mens | pinguin ) echo -n "twee";;
     *) echo -n "een onbepaald aantal";;
  esac
  echo " poten."
```

In het voorbeeld wordt het read commando gebruikt om input in te lezen van het keyboard.

2.5 for

Een for gebruik je om een reeks te doorlopen. Deze reeks kan met getallen zijn of een reeks tekst zijn (gescheiden door een spatie of door een enter).

```
#!/bin/bash
# Functie: Snelle netwerkscan met pings
for teller in {1..254}
do
   ping -n 1 192.168.1.$teller &
done
```

Een for is handig in combinatie met de output van een ander programma. Hiervoor gebruik je backquotes (Alt Gr + µ).

```
#!/bin/bash
# Functie: lijst tonen van bestanden in een directory

directory="/usr/share/backgrounds"

for bestand in `ls $directory`
 do
   echo "Bestand: $bestand"
done
```

Een for kan je (net zoals een while) gebruiken om lijn per lijn een bestand uit te lezen:

```
#!/bin/bash
# Functie: Lijn per lijn het passwd bestand tonen

bestand="/etc/passwd"

for lijn in `cat /etc/passwd`
 do
   echo "$lijn"
 done

Iets uitgebreider met een teller:


#!/bin/bash
# Functie: Lijn per lijn het passwd bestand tonen met teller
teller=0
bestand="/etc/passwd"

for lijn in `cat /etc/passwd`
 do
   echo "Lijn $teller: $lijn "
```

```
   teller=`expr $teller + 1`   # eentje bijtellen  (backquotes!)
 done
```

Output van een commando in een variabele. Opgelet: newlines veranderen in spaties.

```
#!/bin/bash
# Functie:      Tellen aantal bestanden
# Opmerking:    Werkt niet met spaties in bestandsnaam
directory="/usr/share/backgrounds"
lsoutput=`ls $directory`

for bestand in $lsoutput
 do
   echo "Bestand: $bestand"
done

# variabele met ls heeft spaties
echo "In totaal `echo $lsoutput | wc --words` bestanden"

# ls geeft lijnen
echo "In totaal `ls $directory | wc --lines` bestanden"
```

3 BASH BEST PRACTICE

Can you practice what you preach
would you turn the other cheek?
-- TBEP , Where is the love

3.1 Hoofding

De eerste lijn van het script specifieert de soort shell of het programma waarmee het draait. Dit script schrijft zelf een shell script met de juiste hoofding en maakt het script daarna uitvoerbaar:

```
#!/bin/bash
script="script.sh"
echo '#!/bin/bash' > $script
echo "echo \"Hello\" >> $script
chmod +x $script
```

Bij perl wordt de hoofding: #!/usr/bin/perl
Bij python wordt de hoofding: #!/usr/bin/python of #!/usr/bin/env python

```
#!/bin/bash
# Functie:      Kijkt na of iemand een argument meegaf
#               Drukt het eerste argument af of toont een helptekst
if [ $# -lt 1 -o "$1" = "--help" ] ; then
  cat <<-END
          Usage: `basename $0` arg1
          Example: `basename $0` Hallo
END
exit 1
fi
echo "Eerste argument is $1"
```

3.2 Commentaar

In commentaar voorzie je de nodige informatie over het script, de auteur en de licentie. De standaard bash plugin voor bash voorziet bijvoorbeeld volgende hoofding bij een nieuw shell script:

```
#===============================================================
#
#          FILE:  myscript.sh
#
#         USAGE:  ./myscript.sh
#
#   DESCRIPTION:
```

```
#
#       OPTIONS:  ---
#  REQUIREMENTS:  ---
#          BUGS:  ---
#         NOTES:  ---
#        AUTHOR:  Jan Celis (jan.celis@kdg.be)
#       COMPANY:   KdG
#       VERSION:  1.0
#       CREATED:  02/14/12 15:42:08 CET
#      REVISION:  ---
#================================================================
```

3.3 Input validatie

Kijk na of de input klopt die gebruikers ingeven:

Voor het testen van integer getallen, kan er een foutmelding komen wanneer het getal geen integer is. Deze foutmelding vang je op door deze door te sturen naar /dev/null.

```
#!/bin/bash
# Functie: Test of de input een integer getal is
echo "Geef een getal: "
read getal
if [ "$getal" -eq "$getal" ] 2>/dev/null; then
# Geeft normaal integer expression expected
  echo $getal is een getal
else
  echo $getal is geen getal
fi
```

Voor het nakijken van bestanden kan je -f gebruiken, dit kijkt na of het bestand bestaat. De test -s kijkt na of het bestand bestaat én de bestandsgrootte groter is dan 0.

```
#!/bin/bash
# Functie: Toont een niet leeg en bestaand bestand
echo "Geef de bestandsnaam: "
read bestandsnaam
if [ -s "$bestandsnaam" ]; then
  cat $bestandsnaam
else
  echo $bestandsnaam werd niet gevonden
fi
```

Directories kan je nakijken met de optie -d.

```
#!/bin/bash
# Functie: Toont een niet leeg en bestaand bestand
directory="./tmp"
if ! [ -d "$directory" ]; then
  mkdir "./tmp"
fi
```

3.4 Foutafhandeling

3.4.1 Exit code

Een programma geeft exit code 0 als alles juist verloopt.
Een programma geeft een exit code die NIET gelijk is aan nul bij falen.
Je kan aan een falend programma verschillende error codes toekennen met getallen van 1 tot 255. Deze exit code kan je opvragen na het uitvoeren van eender welk programma met de speciale variabele $? In het voorbeeld wordt het programma mkdir uitgevoerd. Als mkdir een foutmelding geeft, eindigt het shell script met exit code 1.

```
#!/bin/bash
# Functie: Aanmaken directory
directory="test"
mkdir $directory
if [ $? != 0 ]] ; then
   echo "$directory aanmaken niet gelukt" >&2
   exit 1
 else "$directory aanmaken gelukt"
   exit 0
fi
```

3.4.2 Foutkanalen

Programma's werken via verschillende kanalen:
STDIN leest input naar een programma en heeft kanaal nummer 0
STDOUT geeft de output van een programma en heeft kanaal nummer 1
STDERR geeft de foutmeldingen van een programma en heeft kanaal nummer 2

```
#!/bin/bash
# Functie:   Wanneer users.csv niet bestaat
#            eindigen met een foutboodschap
input_bestand="users.csv"
if ! [ -f $input_bestand ]; then
  echo "$input_bestand bestaat niet" >&2
  exit 1
fi
```

Volgend script toont het gebruik van kanaal 1 en kanaal 2. Het gebruik van kanaal 1 (>&1) is overbodig omdat het default gebruikt wordt.

```
#!/bin/bash
# Functie:      Nakijken of een directory bestaat
# Naam:   error.sh
directory="test"
if [ -d "$directory" ];
     then
     echo "directory bestaat" >&1
     exit 0
 else
     echo "directory bestaat niet" >&2
     exit 1
fi
```

OUTPUT:

```
jancelis@kdguntu:~$ ./error.sh
directory bestaat niet
jancelis@kdguntu:~$ echo $?
1
jancelis@kdguntu:~$ mkdir test
jancelis@kdguntu:~$ ./error.sh
directory bestaat
jancelis@kdguntu:~$ echo $?
0
```

Ander gebruik van foutkanalen:
a) Je kan ook het foutkanaal STDERR doorsturen naar STDOUT. Dat heet een redirect.

```
mkdir test 2>&1
```

Opmerking: De ampersand geeft aan dat het om het kanaal 1 gaat, moest je de ampersand weglaten, dan maakt bash het bestand met de naam "1" aan.

b) Je kan ook alle foutboodschappen in *the matrix* ;-) laten verdwijnen door deze door te sturen naar /dev/null

```
mkdir test 2>/dev/null
```

c) je kan ook alle kanalen doorsturen als volgt

```
mkdir test &>/dev/null
```

d) of toevoegen aan een bestand

```
mkdir test 2>> errorlog.log
```

3.4.3 Relevante foutberichten

Een foutbericht bevat de naam van de functie of het programma.

```
#!/bin/bash
# Functie: Geeft een foutbericht bij volle schijf
func_df()
{
   err_df="Fout in $0 functie $FUNCNAME:Je schijf is vol"
   df_percent=`df /| tr -s ' ' | cut -d' ' -f5 | tail -n1`
   if [ "$df_percent" = "100%" ]; then
    echo "$err_df" >&2
   fi
}
func_df
```

3.5 Missende executables

We willen nakijken of alle executables aanwezig zijn die je script gebruikt.
Als foutmelding kan je aangeven welk pakket gebruikers nog moeten installeren.

Stel dat we bv de ssh client ssh nodig hebben in ons script, dan zoek je als volgt het pakket waar ssh in staat:

```
jancelis@kdguntu:~$ which ssh
/usr/bin/ssh
jancelis@kdguntu:~$ dpkg -S /usr/bin/ssh
openssh-client: /usr/bin/ssh
jancelis@kdguntu:~$
```

Nakijken of een bestand geïnstalleerd is kan met het programma which, maar efficiënter is om dit te doen via het ingebouwde *command* commando.
Een voorbeeld:

```
#!/bin/bash
error_geen_ssh="Het programma ssh is nodig. Installeren kan met sudo apt-get
install openssh-client"

command -v ssh >/dev/null 2>&1 || { echo >&2 $error_geen_ssh; exit 1; }
```

Je kan ook je script de juiste pakketten laten installeren als niet niet zo is, maar security gewijze gaat niet iedereen graag jou script zomaar root rechten geven om iets te kunnen installeren.

4 BASH FUNCTIES EN TELLEN

There are only 10 types of people in the world
-- those who understand binary,
and those who don't

4.1 Bash functies

4.1.1 Basis functie

Een bash functie moet **gedefinieerd** zijn **voor** je ze gebruikt.

```
#!/bin/bash
# Functie: Declaratie van de function "functie"
#          Oproepen van functie

function functie(){
     echo "Dit is de functie"
}

functie
```

Een functie mag voorafgegaan worden door het keyword "function". Verder definiëren de accolades alle acties die er moeten uitgevoerd worden voor die functie.
Een functie oproepen gebeurt met de functienaam

4.1.2 Functie met argumenten

Bij het oproepen van een functie kan je argumenten meegeven:

```
#!/bin/bash
# Functie: Tonen van het argument in kleur
#          kleuren van output

reset='[0m'    # Vergeet NIET te resetten of alles blijft gekleurd!
rood='[0;31m'

function tooninkleur() {
  echo -e "\e$rood $1 \e$reset"
}

tooninkleur "Hallo"
tooninkleur "Tamelijk rood"
```

```
Output
Hallo
Tamelijk rood
```

Let er op dat de argumenten $1 $2 $3 niet de argumenten van het script zijn, maar WEL de argumenten die met de functie worden meegegeven!

```
#!/bin/bash
# Functie: Oproepen van een functie met argumenten
#          kleuren van output

reset='[0m'
rood='[0;31m'
geel='[1;33m'
blauw='[0;34m'

function tooninkleur() {
  echo -e "\e${1}${2} \e$reset"
  echo "Dit is script heet `basename $0`"
  echo "De functie $FUNCNAME werd opgeroepen met:"
  echo "Kleur (argument 1): $1"  # argument van de functie!
  echo "Tekst (argument 2): $2"  # argument van de functie!
}

tooninkleur $rood "Tomaat"
tooninkleur $geel "Kuiken"
```

```
Output:
Tomaat
Dit is script heet colorscript.sh
De functie tooninkleur werd opgeroepen met:
Kleur (argument 1): [0;31m
Tekst (argument 2): Tomaat
Kuiken
Dit is script heet colorscript_arg.sh
De functie tooninkleur werd opgeroepen met:
Kleur (argument 1): [1;33m
Tekst (argument 2): Kuiken
```

4.1.3 Return waarde van een functie

De return waarde van een functie is in principe alleen maar 0 voor *success* en 1 bij een *fail*. Om strings te "returnen" kan je de output van de functie in een variabele steken. Je kan ook variabelen binnen een functie declareren met local, zodat ze niet globaal gelden.

```
#!/bin/bash
# Functie:  "Return" waarde van een functie is een string

function tooninkleur() {
   local reset='[0m'
   local rood='[0;31m'
   echo -e "\e$rood $1 \e$reset"
}
resultaat=$(tooninkleur)    # of resultaat=`tooninkleur`
echo $resultaat
```

4.1.4 Booleaanse functie

Een booleanse functie geeft 0 (true) of 1 (false) als return waarde.

```
#!/bin/bash

checkexist(){
  #$1=filename
if [ -f "$1" ];
  then
    return 0
  else
    return 1
fi
}

# De vierkante haken gelden als je het "test" commando gebruikt
# Voor deze if is dat niet het geval, dus moet je ze weglaten
if checkexist "/etc/passwd" ; then echo passwd ok; fi
```

4.2 Tellen in bash

Tellen in bash(scripts) kan met volgende operatoren:
+ - / * % ** (plus, min, delen, maal, mod, expon.)

4.2.1 expr

```
#!/bin/bash
a=1
b=2
som=`expr $a + $b`
echo $som
```

4.2.2 let

```
#!/bin/bash
a=1
b=2
let "som=$a+$b"
echo $som
```

4.2.3 builtin bash

```
#!/bin/bash
a=1
b=2
som=$(($a+$b))
echo $som
```

4.2.4 bc

Wanneer je komma getallen nodig hebt, maak je gebruik van bc.
Bc verwacht dat je de getallen piped als input.
Met de *scale* parameter kan je aangeven tot op hoeveel na de komma je wil werken

```
#!/bin/bash
a=1
b=2
echo "$a / $b" | bc
echo "scale=10; $a/$b" | bc
```

```
Output:
0
.5000000000
```

ibase en obase
Met ibase en obase geef je weer wat voor soort getallen je als input (ibase) en/of output (obase) meegeeft:

- A decimaal
- 2 binair
- 16 hexadecimaal

```
#!/bin/bash
# Functie: Conversie binair/decimaal/hexadecimaal met bc
echo "decimaal     ->  binair"
echo "obase=2; $b" | bc
echo "binair       ->  decimaal"
echo "ibase=2; 10101010" | bc
echo "hexadecimaal ->  decimaal"
echo "ibase=16; FF" | bc
echo "decimaal     ->  hexdecimaal"
echo "obase=16; 255" | bc
```

5 PARAMETER SUBSTITUTIE

5.1 Variabele ${var}

Een variabele *username* declareer je zonder spaties en zonder dollar teken en bij voorkeur in kleine letters zodat er geen conflict is met een bestaande systeemvariabele. Voor de duidelijkheid mag je *$username* als *${username}* schrijven.

```
#!/bin/bash
 # Functie:      Gebruiken van een variabele

 username=`whoami` # met backquotes, de output van whoami wordt de waarde
 echo "Dit script wordt uitgevoerd door ${username}
```

```
Output:
Dit script wordt uitgevoerd door user1
```

De accolades zijn zelfs noodzakelijk wanneer je een variabele zonder spaties in een string wil plaatsen

```
#!/bin/bash
var="hallo"
echo "$varmetdezoo"      # werkt niet!
echo "${var}metdezoo"    # werkt wel
```

```
Output:

hallometdezoo
```

5.2 Default waarde ${var:-default}

Een niet gedeclareerde variabele kan je een default waarde toekennen

```
#!/bin/bash
 # Functie:      Gebruiken van een variabele met default waarde

 echo "Dit script wordt uitgevoerd door ${username:-`whoami`}
```

Handige bij deze default waarden is dat je ze kan gebruiken voor ontbrekende commandline argumenten

```
 #!/bin/bash
 # Functie:      Zoeken naar grote bestanden, default groter dan 10 MB
 # Naam:         findbig.sh
 # Argumenten:   Argument $1 MAG meegegeven worden
 #               Als $1 leeg is, krijgt het een default waarde

 default_waarde="10"
 size=${1:-$default_waarde}
 find . -iname "*"  -size "+${size}M"
```

```
Output:
 jancelis@kdguntu~$ ./findbig.sh
 /home/jancelis/Downloads/iso/slitaz-cooking.iso
 /home/jancelis/Downloads/iso/ubuntu-13.04.1-server-amd64.iso
```

5.3 Foutmeldingen ${var?error_message}

Lege variabelen kan je automatisch een error message toekennen én laten eindigen met exit code 1

```
#!/bin/bash
 # Functie:      Grote bestanden vinden
 # Argumenten:   Argument $1 MOET meegegeven worden
#                Als $1 leeg is eindigt het script met exit code 1
# Naam:          findbig.sh

size=${1?"Usage: $0 ARGUMENT"}

# Script komt enkel hier wanneer $1 werd ingevuld
find /home/jancelis -iname "*"  -size "+${size}M"
```

Je kan dit ook als volgt schrijven:

```
#!/bin/bash
 # Functie:      Grote bestanden vinden
 # Argumenten:   Argument $1 MOET meegegeven worden
 #               Als $1 leeg is eindigt het script met exit code 1
 # Naam:         findbig.sh

 # De dubbelpunt : geeft een "true", en verhindert dat het script
 # de variabele $1 als commando probeert uit te voeren
: ${1?"Usage: `basename $0` ARGUMENT"}

 # Script komt enkel hier wanneer $1 werd ingevuld
 find . -iname "*"  -size "+${1}M"
```

```
 Output:
```

```
jancelis@kdguntu~$ ./findbig.sh
./findbig.sh: line 9: 1: Usage: ./findbig.sh ARGUMENT
```

5.4 Replace

Algemene vorm: `${string/vervang_deze_string/door_deze_string}`

```
#!/bin/bash
# Functie: Vervangen van een string in een url

 url="http://www.kdg.be/index.html"
echo String ${url}

echo Vervang 1 keer kdg door student  ${url/kdg/student}
echo Vervang alle keren ht door f     ${url//ht/f}

echo Vervang begin met http door ftp  ${url/#http/ftp}
echo Vervang einde met html met aspx  ${url/%html/aspx}
```

```
Output:
String  http://www.kdg.be/index.html
Vervang 1 keer kdg door student  http://www.student.be/index.html
Vervang alle keren ht door f     ftp://www.kdg.be/index.fml
Vervang begin met http door ftp  ftp://www.kdg.be/index.html
Vervang einde met html met aspx  http://www.kdg.be/index.aspx
```

Opmerking:
Je kan een replace ook gebruiken om een string te verwijderen door de vervangterm leeg te laten

```
#!/bin/bash
 echo Verwijder 1 keer kdg ${url/kdg}
 echo Verwijder alle keren ht ${url//ht}
```

```
Output:
Verwijder 1 keer kdg         http://www..be/index.html
Verwijdere alle keren ht     tp://www.kdg.be/index.ml
```

5.5 Substring

Algemene vorm: `${string:positie:lengte}`

```
#!/bin/bash
url="http://www.kdg.be/index.html"
echo String ${url}

echo Eerste 7 karakters wegknippen ${url:7}
# Haakjes of spatie is escape van positie
echo Laatste 4 karakters ${url: -4}
echo Laatste 4 karakters ${url:(-4)}
echo Eerste 4 karakters  ${url:0:4}
echo Karakter 8 tot 18   ${url:7:10}
```

```
Output:
Alles vanaf karakter 8 krabt de krollen van de trap
Alles tot het 3de karakter De
Alles vanaf karakter 4 tot 13 kat krabt
Alles vanaf het 4 de laatste karakter trap
```

5.6 Trim

`${string#substring}` verwijdert de kortste match substring vooraan in de string.
Dit principe om te matchen met de kortste match noemen we ook wel **ungreedy**
`${string##substring}` verwijdert de langste match substring vooraan string
Dit principe om te matchen met de langste match noemen we ook wel **greedy** (gretig zijn).

```
#!/bin/bash
# Functie: afkappen string vooraan en achteraan, greedy en ungreedy

url="http://www.kdg.be/index.html"
echo $url
echo korste substring http:// vooraan: ${url#http://}
echo langste substring http\*. vooraan: ${url##http*.}
echo
echo korste substring non-greedy .\* achteraan: ${url%.*}
echo langste substring greedy .\* achteraan: ${url%%.*}
```

```
Output:
http://www.kdg.be/index.html
korste substring http:// vooraan: www.kdg.be/index.html
langste substring http*. vooraan: html

korste substring non-greedy .* achteraan: http://www.kdg.be/index
langste substring greedy .* achteraan: http://www
```

5.7 Lengte

Doel: Lengte van de string berekenen

```
url="http://www.kdg.be/index.html"
echo Lengte van de string: ${#url}
```

```
Output: Lengte van de string:  28
```

5.8 Lower/upper

Doel: Alles omzetten naar lowercase/uppercase

```
url="http://www.kdg.be/index.html"
echo Alles uppercase: ${url^^}
echo Alles lowercase: ${url,,}
```

Dit kan ook met het programma tr (translate). Je kan `tr [:lower:] [:upper:]` gebruiken

```
echo 'Alles uppercase: http://www.kdg.be/index.html' | tr a-z A-Z
```

```
Output:
Alles uppercase: HTTP://WWW.KDG.BE/INDEX.HTML
Alles lowercase: http://www.kdg.be/index.html
```

6 REGULIERE EXPRESSIES

Today's Special
~~Beer $ 1.50~~
Beer ^1\.50$

--Geek&poke

Reguliere expressies kan je gebruiken om te zoeken naar een bepaald patroon in een tekst. Bash volgt de POSIX standaard voor reguliere expressies. Deze worden bij IEEE 1003 gedefinieerd als BRE (Basic Regular Expressions) en ERE (Extended Regular Expressions).

6.1 Basis regex scripts

6.1.1 =~

De speciale operator =~ kijkt met regex na of een patroon voorkomt in een string. De operator is gelijkaardig bij perl.

```
#!/bin/bash
content="Karel de Grote-Hogeschool, Nationalestraat 5,B-2000 Antwerpen"

# Regex matching met =~ operator tussen [[ dubbele vierkante haken ]].
if [[ $content =~ B-[0-9]{4} ]]
  then
  echo "postnummer gevonden"
fi
```

```
Output: postnummer gevonden
```

Een verbetering van het script, krijgen we, wanneer we enkel een output geven wanneer het nodig is. Wanneer er geen match is, eindigen we met een exit code 1, om aan te geven dat er een fout was.

```
#!/bin/bash
content="Karel de Grote-Hogeschool, Nationalestraat 5,B-20 Antwerpen"
if [[ $content =~ B-[0-9]{4} ]]
  then
    echo "postnummer gevonden" ;  exit 0
  else
    echo "postnummer niet gevonden" >&2 ; exit 1
fi
```

```
Output:
jancelis@kdguntu~$ ./zoekpostnummer.sh
jancelis@kdguntu~$ echo $?
1
```

Bij gebruik van een variabele met een regex, mag de regex niet tussen quotes staan (dit was bij oudere bash versies (<4.0) wel zo)!:

```
#!/bin/bash
content="Karel de Grote-Hogeschool, Nationalestraat 5,B-2000 Antwerpen"
regex="B-[0-9]{4}"
 if [[ $content =~ $regex ]]
  then
    echo "postnummer gevonden" ;  exit 0
  else
    echo "postnummer niet gevonden" >&2 ; exit 1
fi
```

```
Output: postnummer gevonden
```

Weergeven van het gevonden resultaat kan met de speciale variabele ${BATCH_REMATCH}. Daarin komt een array met alle matches die gevonden werden. Element 0 van de array bevat de hele match, de rest van de array kan regex groepen bevatten (zie later).

```
#!/bin/bash
# Functie: Regex voor een postnummer
content="Karel de Grote-Hogeschool, Nationalestraat 5,B-2000 Antwerpen"
regex="B-[0-9]{4}"

[[ $content =~ $regex ]]
echo "${BASH_REMATCH[0]}"
```

```
Output: B-2000
```

6.2 Regex speciale karakters

Door het gebruik van speciale karakters in een regex, kunnen we complexere zoekstructuren maken Opgelet: Wanneer er één van deze karakters voorkomt in de zoekterm, moet je deze escapen met backslash. Dit gaat om de karakters \ . * [+ {] }. Binnen vierkante haakjes moet er bij bash niet escaped worden. Een speciaal karakter zet je wel éérst tussen de vierkante haakjes.

6.2.1 Regex van een postnummer

Hier bouwen we stelselmatig een regex op voor een postnummer met landcode.
We weten bijvoorbeeld dat het postnummer altijd een decimaal getal is. Dat kunnen we aangeven met [0-9] of [[:digit:]]:
`regex="[[:digit:]][[:digit:]][[:digit:]][[:digit:]]"`
Om herhaling te vermijden geven we met accolades aan hoeveel keer we een "*digit*" verwachten.
`regex="[[:digit:]]{4}"`

Wanneer we deze regex iets willen uitbreiden, zodat ook andere landcodes kunnen gedetecteerd worden, kunnen we 'B' als een letter aangeven met [[:alpha:]]
`regex="[[:alpha:]]-[[:digit:]]{4}"`

Aangezien landcodes 1 of 2 karakters kunnen hebben, geven we bij de eerste `[[:alpha:]]` aan dat deze 1 of 2 keer kan voorkomen. Dat doen we met accolades.
`regex="[[:alpha:]]{1,2}-[[:digit:]]{4}"`

Hetzelfde kunnen we met de nummers doen. We gaan er van uit dat een postcode van 4 tot 6 karakters kan zijn \d {4,6} betekent dus van 4 tot 6 \d karakters:
`regex="[[:alpha:]]{1,2}-[[:digit:]]{4,6}"`
Wat we ook nog kunnen toevoegen is dat achter de postcode er altijd een spatie staat. Het speciale karakter voor een spatie, tab of newline is [[:space:]]. [[:space:]] komt eigenlijk overeen met [\t\n\r\f\v] (spatie, tab, newline, carriagereturn, formfeed, verticalfeed).
`regex="[[:alpha:]]{1,2}-[[:digit:]]{4,6}[[:space:]]"`

Soms wordt het streepje niet geschreven tussen de landcode en de postcode maar wel een spatie. Deze logische OR kunnen we weergeven met vierkante haken [-]
`regex="[[:alpha:]]{1,2}[- ][[:digit:]]{4,6}[[:space:]]"`

Deze regex mag je ook schrijven als:
`regex="[a-zA-Z]{1,2}[- ][0-9]{4,6} "`

```
#!/bin/bash
# Functie: Regex voor een email adres
content="Karel de Grote-Hogeschool, Nationalestraat 5,B-2000 Antwerpen"
regex="[[:alpha:]]{1,2}[- ][[:digit:]]{4,6}[[:space:]]"
#regex="[a-zA-Z]{1,2}[- ][0-9]{4,6} "

[[ $content =~ $regex ]]
echo "${BASH_REMATCH[0]}"
```

```
Output:  B-2000
```

6.2.2 Regex van een email adres

Hier bouwen we stelselmatig een regex op om een een email adres te matchen. De karakters van een email adres kunnen een onbepaalde lengte hebben (minimum 1) en schrijven we als [[:alnum:]]+

Dit geeft:

jan . *celis* @ *kdg* . *be*

```
[[:alnum:]]+ .      [[:alnum:]]+   @       [[:alnum:]]+ .  [[:alpha:]]+
```

Het eerste stuk kan ook zonder punt zijn, dus dat plaatsen we optioneel:

jancelis @ *kdg* . *be*

```
([[:alnum:]]+.){0,2} [[:alnum:]]+  @  [[:alnum:]]+.  [[:alpha:]]+
```

Het tweede deel kan ook uit subdomeinen bestaan:

jan . *celis* @ *student* . *kdg* . *be*

```
([[:alnum:]]+.){0,2} [[:alnum:]]+  @ ([[:alnum:]]+.){1,3}  [[:alpha:]]+
```

Het laatste domein deel is van 2 tot 3 karakters (zonder cijfers):

jan . *celis* @ *student* . *kdg* . *be*

```
([[:alnum:]]+.){0,2} [[:alnum:]]+  @ ([[:alnum:]]+.){1,3} [[:alpha:]]{2,3}
```

In een bash script geeft dit:

```
#!/bin/bash
# Functie: eenvoudige regex voor email adres

content="jan.celis@kdg.be"

regex="([[:alnum:]]+.){0,2}[[:alnum:]]+@[[:alnum:]]+(.[[:alnum:]]+){0,2}.
[[:alpha:]{2,3}"

[[ $content =~ $regex ]]
echo "${BASH_REMATCH[0]}"
```

```
Output:  jan.celis@kdg.be
```

Deze oplossing is niet volledig (underscores zijn bijvoorbeeld ook toegelaten in email adressen, maar dient als voorbeeld van het gebruik van reguliere expressies. Verder worden .vlaanderen en .brussels niet ondersteund.

De echte RFC 822 email regex is ongeveer een volledige A4!

6.2.3 Simulatie van grep

Volgende script gaat het eerste argument gebruiken als zoekterm. Het is dus een eenvoudige vorm van het unix commando grep. Wanneer er niets gevonden wordt, toont het script niets en geeft het een exit code 1 (zoals bij de echte grep).

```
#!/bin/bash
# Functie:      grep simuleren met bash regex
# Naam:         grep.sh
# Argumenten:   $1 regex
#               $2 filename

if [ -z "$2" ] || [ "$1" = "--help" ]; then
  echo "Usage: `basename $0` [PATTERN] [FILE]"
  exit 1
fi
regex="${1}"
bmatch=false
OLD_IFS="$IFS"                  # oude separator bewaren
IFS=$'\n'                       # \n als separator

for content in `cat ${2}`
  do
    [[ $content =~ $regex ]]
    match="${BASH_REMATCH[0]}"
    if [ -n "${match}" ]; then
       echo ${content}
      bmatch=true
    fi
done
IFS="$OLD_IFS"                  # terug oude \n \t en spatie als IFS
if ! $bmatch ; then exit 1      # geen match geeft exit 1
```

```
Output:
jancelis@kdguntu~$ ./grep.sh test grep.sh
jancelis@kdguntu~$ echo $?
1
jancelis@kdguntu~$ ./grep.sh bin grep.sh
#!/bin/bash
jancelis@kdguntu~$ echo $?
0
```

6.2.4 Leesbaarheid regex

Verspreiden over meerdere lijnen kan de leesbaarheid vergroten. Bij bash kan je de lijn afsluiten met een backslash om verder te gaan op de volgende lijn. Het is echter niet mogelijk om ook nog commentaar achter elke lijn te schrijven

```
#!/bin/bash
# Functie: regex van een email adres

content="jan.celis@kdg.be"
regex="([[:alnum:]]+.){0,2}\
[[:alnum:]]+\
@\
[[:alnum:]]+\
(.[[:alnum:]]+){0,2}\
.[[:alpha:]{2,3}"

[[ $content =~ $regex ]]
echo "${BASH_REMATCH[0]}"
```

6.3 Regex group

Omdat je dikwijls maar een gedeelte van de zoekterm nodig hebt in je programma, kan je gebruik maken van grouping. Je geeft met haakjes aan welk gedeelte van de regex je wil opvragen.

```
#!/bin/bash
# Functie: Voorbeeld gebruik regex groups

content="Karel de Grote-Hogeschool, Nationalestraat 5,B-2000 Antwerpen"

regex="([- a-zA-Z]+), ([[:alpha:]]+) ([[:digit:]]+),(.*) (.*)"
[[ $content =~ $regex ]]
echo "Naam: ${BASH_REMATCH[1]}"
echo "Straat: ${BASH_REMATCH[2]} Nr: ${BASH_REMATCH[3]}"
echo "Postcode: ${BASH_REMATCH[4]} Stad: ${BASH_REMATCH[5]}"
```

```
Output:

Naam: Karel de Grote-Hogeschool
Straat:  Nationalestraat  Nr: 5
Postcode: 2000 Stad:  Antwerpen
```

6.4 Regex en unicode

Bash ondersteunt geen unicode voor regex. Hogere talen zoals python, java en .NET zullen dit wel doen. Dit kan bijvoorbeeld een probleem zijn wanneer je ook arabische of chinese cijfers als [[:digit:]] wil gebruiken, of bestandsnamen hebt die niet alleen uit ASCII karakters bestaan.

6.5 Regex greedy ungreedy

In hogere talen kan je kiezen of een regex greedy (gretig) is of niet. Wanneer deze greedy wordt gebruikt zal de regex zoveel mogelijk proberen te matchen. Ungreedy is de match zo kort mogelijk

Om binnen parameter substitution ook gebruik te maken van extended regex, schakel je de optie extglob in de shell aan. Dat kan met het commando: shopt -q -s extglob

```
#!/bin/bash
content="Karel de Grote-Hogeschool, Nationalestraat 5,B-2000 Antwerpen"
shopt -q -s extglob
content2=${content#*+([[:digit:]])}
content=${content##*+([[:digit:]])}
shopt -q -u extglob
echo "ungreedy: $content2"
echo "greedy: $content"
```

```
Output:
ungreedy: ,B-2000 Antwerpen
greedy:  Antwerpen
```

7 COMPILEREN

Someday somebody has got to decide whether the typewriter is the machine,
or the person who operates it.

7.1 Compileren met gcc/g++

Doel: gcc en g++ zijn de unix compilers voor respectievelijk C en C++ programma's.
In Linux mag je zowel cc (compiler controller) als gcc (GNU cc) gebruiken als programmanaam.
Vorm: `gcc/g++ [opties] [bestanden]`

Uitleg:

Extensies	*Extensies moeten juist zijn!*
.c of .C	Compileren van C source (compileren met gcc)
.cc of .cxx	Compileren van C++ source (compileren met g++)

Opties gcc/g++	*Uitleg*
-o *file*	Output (executable) wordt weggegschreven in *file*
-c	Zonder te linken naar libraries
-l*library*	Gelinkt aan *library* . (librarynaam plakt tegen de L !)
-include *file*	Gebruikt variabelen/constanten uit een *inc bestand*
-g	Met debugging (gebruikt door bv *gdb* debugger)

Voorbeeld:

Compileren van een standaard c programma.

```
#include <stdio.h>
 int main() {
 printf("Hello World/n");
 return (0); }
```

```
kdguntu@kdguntu:~$ gcc -o programma programma.c
kdguntu@kdguntu:~$ ./programma
Hello World
kdguntu@kdguntu:~$
```

```
#include <iostream>
#include "functions.h"

void print_hello(){
   std::cout << "Hello World";
}
```

```
kdguntu@kdguntu:~$ g++ -o programma programma.cpp
kdguntu@kdguntu:~$ ./programma
Hello World
kdguntu@kdguntu:~$
```

7.2 Compileren met make

Het programa make gebruikt een bestand (een Makefile), waarin alle compile opties voor een programma zijn opgenomen.

Als voorbeeld compileren we het c++ programma helloworld vanuit de sources main.cpp, helloworld.cpp en factorial.cpp. Deze bestanden zetten we in de directory src.

Best gebruik je een aparte directory voor elk project.

main.cpp

```
#include <iostream>
#include "functions.h"

int main(){
    print_hello();
    std::cout << std::endl;
    std::cout << "De faculteit van 10 is " << factorial(10) <<
std::endl;
    return 0;
}
```

helloworld.cpp

```
#include <iostream>
#include "functions.h"

void print_hello(){
   std::cout << "Hello World!";
}
```

factorial.cpp

```
#include "functions.h"

int factorial(int n){
    if(n!=1){
    return(n * factorial(n-1));
    }
    else return 1;
}
```

functions.h

```
void print_hello();
int factorial(int n);
```

7.2.1 Make

Installeren: sudo apt-get install make

Doel: Het programma make zoekt naar een bestand met de naam "makefile" in de directory, en zal dit uitvoeren (mogelijk met opties)

De compiler verzamelt de source files en geeft een object bestand. De linker maakt van de object bestanden een binair bestand (een executable). Opgelet!
In Linux eindigen niet alle binaire bestanden op de extensie .exe!

Om dit voorbeeld met de hand te compileren, moet je volgend commando uitvoeren:

```
g++ main.cpp helloworld.cpp factorial.cpp -o helloworld
```

Hierbij is helloworld de naam van de executable.

7.2.2 Een basis Makefile

De algemene syntax van een makefile is als volgt:

```
target: dependencies
    [tab] systeemcommando's
```

Bij ons voorbeeld komt er in de makefile volgende inhoud:

```
all:
    g++ main.cpp helloworld.cpp factorial.cpp -o helloworld
```

Om deze makefile uit te voeren (en dus te compileren) doe je:

```
kdguntu@kdguntu~$ make -f Makefile -1
```

Je ziet dat bij dit voorbeeld de target "all" heet. Dat is de default target voor makefiles. Het make programma zal "all" uitvoeren wanneer er geen target werd opgegeven.

Er zijn geen dependencies voor de target "all" gedefinieerd, dus de systeem commando's worden uitgevoerd.

7.2.3 Met dependencies

Soms is het handig om verschillende targets te definiëren. Wanneer je één bestand verandert in je project, moet je niet het hele project hercompileren, maar enkel de veranderde delen.

Voorbeeld:

```
all: helloworld

helloworld: main.o factorial.o helloworld.o
    g++ main.o factorial.o helloworld.o -o helloworld

main.o: src/main.cpp
    g++ -c src/main.cpp

factorial.o: src/factorial.cpp
    g++ -c src/factorial.cpp

helloworld.o: src/helloworld.cpp
    g++ -c src/helloworld.cpp

clean:
    rm -rf *o helloworld
```

De target "all" bestaat hier enkel uit dependencies, maar heeft zelf geen systeemcommando's. Om juist "all" uit te voeren, moeten alle dependencies uitgevoerd worden.
De opdeling in verschillende targets bespaart tijd wanneer een target al gecompileerd is. Wanneer bijvoorbeeld enkel factorial.cpp aangepast is, moet enkel dat gedeelte opnieuw gecompileerd worden.

De target clean bevat de commando's om alle objectbestanden en executables te verwijderen.

7.2.4 Variabelen en commentaar

Je kan ook variabelen gebruiken in een makefile. Dat is bijvoorbeeld handig om compiler opties aan te passen.

```
# Dit is commentaar
# De variabele CC bevat de compiler die gebruikt zal worden
CC=g++
# De variabele CFLAGS bevat de compileopties
CFLAGS=-c -Wall

all: helloworld

helloworld: main.o factorial.o helloworld.o
    $(CC) main.o factorial.o helloworld.o -o helloworld
```

```
main.o: main.cpp
    $(CC) $(CFLAGS) main.cpp

factorial.o: factorial.cpp
    $(CC) $(CFLAGS) factorial.cpp

helloworld.o: helloworld.cpp
    $(CC) $(CFLAGS) helloworld.cpp

clean:
    rm -rf *o helloworld
```

```
# Met variabelen voor alle bestanden
CC=g++
CFLAGS=-c -Wall
SOURCES=main.cpp helloworld.cpp factorial.cpp
OBJECTS=$(SOURCES:.cpp=.o)
EXECUTABLE=helloworld

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
    $(CC) $(OBJECTS) -o $@

.cpp.o:
    $(CC) $(CFLAGS) $< -o $@
```

7.3 Manpage

Volgens de debian policy heeft elke executable die je verdeelt een man page nodig. Deze kan je genereren met gmanedit (sudo apt-get install gmanedit).

Een manpage heeft als extensie een sectienummer. De sectienummer wordt bepaald door het soort programma. Mogelijke man page secties:

Nummer	Sectie	Uitleg
1	User command	Uitvoerbare commando's en scripts
2	System calls	Kernel functies
3	Library calls	Functies uit systeembibliotheken
4	Special files	Meestal bestanden in /dev
5	File formats	Bijvoorbeeld formaat /etc/passwd
6	Games	Spelletjes of andere frivoliteiten
7	Macro packages	Bijvoorbeeld man macro's
8	System administration	Programma's die enkel door root worden uitgevoerd
9	Kernel routines	Niet-standaard aanroepen en interne kernel functies

Voorbeeld man pagina:

```
 .TH helloworld 1 "February 25, 2014" "" "helloworld"

 .SH NAME
 helloworld \- program for displaying Hello World on the screen

 .SH SYNOPSIS
 .B helloworld
 .RI [ no options ]
 .br

 .SH DESCRIPTION
 This manual page explains how helloworld works. The program displays a
friendly Hello World message on the screen and exits.
 .B helloworld
 program. This program shows Hello World
 .PP
 \fBhelloworld\fP is for showing Hello World

 .SH EXAMPLE
 helloworld
 Hello World
```

7.4 Autoconf en automake

In de linux wereld is er een bijna defacto standaard om voor een compilatie het commando ./configure && make && make install uit te voeren.

Maak volgende structuur aan voor je source:

Source files in src/

Documentatie in doc/

Man pages in man/

Scripts in scripts/ (Scripts is alles wat niet gecompileerd moet worden)

Voorbeelden in examples/

src helloworld.cpp

```
kdguntu@kdguntu:~$ ls -R
.:
doc  man  src

./doc:
INSTALL

./man:
helloworld.1

./src:
helloworld.cpp

kdguntu@kdguntu:~$ autoscan
kdguntu@kdguntu:~$ ls
autoscan.log  configure.scan  doc  man  src
kdguntu@kdguntu:~$ mv configure.scan configure.ac
```

Editeer volgende lijn in configure.ac

```
AC_INIT([FULL-PACKAGE-NAME], [VERSION], [BUG-REPORT-ADDRESS])
```

naar de gegevens van je eigen pakket

```
AC_INIT([helloworld], [1.0], [kdguntu@gmail.com])
```

Nu kan je met autoconf een configure script genereren:

```
kdguntu@kdguntu:~$ autoconf
kdguntu@kdguntu:~$ ls
autom4te.cache  autoscan.log  configure  configure.ac  doc  man
src
```

Maak nu een Makefile.am bestand aan in de hoofddirectory. Je mag/kan ook in elke subdirectory een Makefile.am bestand plaatsen, maar dat maakt alles minder overzichtelijk.
Bij een standaard GNU project horen volgende bestanden:

```
kdguntu@kdguntu:~$touch doc/NEWS doc/README doc/AUTHORS doc/ChangeLog
```

Gebruik je niet de standaard GNU vorm, dan geef je dit aan door in Makefile.am volgende lijn toe te voegen:

```
AUTOMAKE_OPTIONS = foreign
```

Voorbeeld Makefile.am configuratie

```
SUBDIRS = doc man src


bin_PROGRAMS = helloworld
helloworld_SOURCES = helloworld.cpp
man_MANS = helloworld.1

docdir = $(datadir)/doc/@PACKAGE@
doc_DATA = INSTALL README
```

We definiëren nu enkele macro's in configure.ac om aan autoconf te zeggen dat de Makefiles moeten gemaakt worden na de ./configure

```
kdguntu@kdguntu:~$ vi configure.ac
```

Net na AC_INIT(), initializeren we automake:

AM_INIT_AUTOMAKE(helloworld, 1.0)

Daarna laten we autoconf een script schrijven dat Makefiles maakt voor alle directories.

AC_OUTPUT(Makefile src/Makefile doc/Makefile man/Makefile)

Met het programma aclocal worden automatisch alle scripts aangemaakt:

```
kdguntu@kdguntu:~$ aclocal
```

Dit maakt het bestand aclocal.m4 met macros voor automake , bv. AM_INIT_AUTOMAKE.

```
kdguntu@kdguntu:~$ automake
```

Automake leest configure.ac en de top-level Makefile.am en maakt voor elke Makefile.am een Makefile.in.

Nu kan autoconf het configure script schrijven:

```
kdguntu@kdguntu:~$ autoconf
```

Verdere fine-tuning

Wanneer ze zelf nog checks wil toevoegen aan het configure script, dan kan je de shell code hiervoor schrijven in configure.ac (voor OUTPUT commando's), en dan opnieuw autoconf draaien.

8 REGEX IN VERSCHILLENDE TALEN

In dit hoofdstuk vergelijken we het gebruik van reguliere expressies met **groups** bij verschillende talen.

Als voorbeeld downloaden we een website (hier standaard een webserver op onze eigen machine) en halen met een eenvoudige regex de *h1* hoofding uit de source. Door het gebruik van groups kunnen we enkel het tekstgedeelte opvragen.

Je zal zien dat de regex en groupdefinitie identiek blijft bij alle talen: **'.*<h1>(.*)</h1>.***
Het unix commando grep ondersteunt geen regex groups maar wel regex bv:

```
echo "<html><h1>It works</h1></html>" |  grep -Po '<h1>(.*)</h1>'
```

8.1 Regex met python

```
#!/usr/bin/env python
# Python <h1> informatie </h1> uit website halen
import re
from urllib import urlopen
site="127.0.0.1"
regex='.*<h1>(.*)</h1>.*'
content= urlopen("http://"+site)
text= content.read()
regextext = re.search(regex,text)
print regextext.group(1)
```

8.2 .Regex met perl

```
#!/usr/bin/perl
# Perl <h1> informatie </h1> uit website halen
use LWP::Simple;
$site="http://127.0.0.1";
$tmp="tmp.html";
$regex='.*<h1>(.*)</h1>.*';
my $content = get($site);
if ( $content =~ $regex) {
  print "$1\n";
}
```

8.3 Regex met bash

```
#!/bin/bash
# Bash <h1> informatie </h1> uit website halen
site="127.0.0.1"
# tmp="tmp.html";content=`wget --quiet -O $tmp $site` # alternatief
regex='.*<h1>(.*)</h1>.*'
 content=`exec 3<>/dev/tcp/$site/80;
          echo -e "GET / HTTP/1.0\n" >&3;cat <&3`
[[ $content =~ $regex ]]
match="${BASH_REMATCH[1]}"
echo $match
```

8.4 Regex met sed

```
#!/bin/bash
# Sed <h1> informatie </h1> uit website halen
site="127.0.0.1"
content=`curl --silent $site`
regex='.*<h1>(.*)<\/h1>.*'
echo $content|sed -r "s/$regex/\1/"
```

8.5 .Regex met c sharp

```
// C sharp <h1> informatie </h1> uit website halen
using System;
using System.Net;
using System.Text.RegularExpressions;
public class GetSite
{
   public static void Main()
   {
      string site="http://127.0.0.1";
      Regex r1 = new Regex(@".*<h1>(.*)</h1>.*");
      WebClient client = new WebClient();
      string content = client.DownloadString(site);
      Match match = r1.Match(content);
      if (match.Success)
         {
            Console.WriteLine(match.Groups[1].Value);
        }
   }
}
```

8.6 Regex met java

```
import java.io.*;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.regex.*;

public class GetSite {
  public static void main(String[] args) {
    try {
         URL site = new URL("http://127.0.0.1");
         BufferedReader reader = new BufferedReader
                              (new InputStreamReader(site.openStream()));
         String line;
         while ((line = reader.readLine()) != null) {
            Pattern p = Pattern.compile(".*<h1>(.*)</h1>.*");
             Matcher m = p.matcher(line);
            while (m.find()){
              System.out.println(m.group(1));
            }
         }
         reader.close();
        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

8.7 .Vervangen van een parameter met sed

Het unix commando sed ondersteunt, net als grep, reguliere expressies.

Het volgende voorbeeld past in het apache configuratiebestand de poort aan van poort 80 naar poort 8080 zodat de webserver op poort 8080 luistert:

```
#!/bin/bash
 bestand="ports.conf"

function replaceline(){
# $1 filename
# $2 searchregex
# $3 replaceterm
sed -i "s/${2}/${3}/" ${1}
}

replaceline $bestand "Listen 80" "Listen 8080"
```

8.8 .Vervangen van parameter met bash

Het volgende voorbeeld past in het apache configuratiebestand de poort aan van poort 80 naar poort 8080 zodat de webserver op poort 8080 luistert:

```
#!/bin/bash
 bestand="ports.conf"

function replaceline(){
# $1 filename
# $2 searchregex
# $3 replaceterm
OLD_IFS="$IFS"
IFS=$'\n'
replaceregex="$2"
for lijn in `cat ${1}`
  do
     if [[ $lijn =~ $replaceregex ]] ; then
       lijn=${lijn/$replaceregex/"$3"}
     fi
  echo ${lijn} >> ${1}
done
IFS="$OLD_IFS"
}
replaceline $bestand "Listen 80 " "Listen 8080"
```

9 BASH PARAMETER EXPANSION

Functie	Inputstring s	Bash	Output
Weergeven	ABC123abc123	${s}	ABC123abc123
Weergeven UPPERCASE	ABC123abc123	${s^^}	ABC123ABC123
Weergeven lowercase	ABC123abc123	${s,,}	abc123abc123
Weergeven Lengte	ABC123abc123	${#s}	12
Weergeven **laatste 4** karakters	ABC123ab**c123**	${s: -4}	c123
Weergeven **eerste 4** karakters	**ABC1**23abc123	${s:0:4}	ABC1
Weergeven karakter **4 tot 10**	ABC**123abc**123	${s:3:6}	123abc
Verwijder **eerste 7** karakters	~~**ABC123a**~~bc123	${s:7}	bc123
Verwijder **laatste 4** karakters	ABC123ab~~**c123**~~	${s::-4}	ABC123ab
Verwijder **eerste 2** en **laatste 3** karakters	~~**AB**~~C123abc~~**123**~~	${s:2:-3}	C123abc
Verwijder substring **A*12** vooraan ungreedy	~~**ABC12**~~3abc123	${s#A*12}	3abc123
Verwijder substring **A*12** vooraan greedy	~~**ABC123abc12**~~3	${s##A*12}	3
Verwijder substring **123*** achteraan ungreedy	ABC123abc~~**123**~~	${s%123*}	ABC123abc
Verwijder substring **123*** achteraan greedy	ABC~~**123abc123**~~	${s%%123*}	ABC
Verwijder 1 keer substring **123**	ABC~~**123**~~abc123	${s/123}	ABCabc123
Verwijder overal substring **123**	ABC~~**123**~~abc~~**123**~~	${s//123}	ABCabc
Verwijder **alle letters**	~~**ABC**~~123~~**abc**~~123	${s//[a-z]}	123123
Vervang één keer **123** door **abc**	ABC**123**abc123	${s/123/abc}	ABCabcabc123
Vervang overal **123** door **abc**	ABC**123**abc**123**	${s//123/abc}	ABCabcabcabc
Vervang begin **ABC** door **123**	**ABC**123abc123	${s/#ABC/123}	123123abc123
Vervang einde met **123** door **abc**	ABC123abc123	${s/%123/abc}	ABC123abcabc

10 REGEX CHEAT SHEET

Teken(s)	Betekenis	Voorbeeld
^	Patroon moet aan het begin van een lijn staan	^kat matcht een string die met kat begint
$	Patroon moet aan het einde van een lijn staan	kat$ matcht een string die eindigt met kat
.	Eén keer eender welk karakter	ka. matcht met kat kap kam maar niet met kast
?	vorige item moet nul of één keer voorkomen	ka? matcht met k of ka
*	vorige item moet nul of meer keer voorkomen	ka* matcht met k of ka of kaa of kaaaaaa
+	vorige item moet één of meer keer voorkomen	ka+ mathed met ka of kaa of kaaaaa maar niet met k
.*	nul of meer keer eender welk karaker	matcht eender welke string (ook van lengte nul)
[]	komt overeen met één van de karakters	[kK]at matcht kat en Kat
[^]	komt NIET overeen met 1 van de karakter	[^k]at matcht Kat, nat, gat,... maar NIET kat
[-]	alfabetische /numerische range van karakters	[a-z] matcht met kat, nat, .. maar niet met hoofdletters
()	Groepering van substring voor meta karakters	k(at)? matcht k of kat maar niet ka of kast
{n}	Exact aantal keer dat iets moet voorkomen	ka{2} matcht alleen maar kaa
{n,}	Minimum aantal keer dat iets moet voorkomen	ka{2,} matcht kaa, kaaa, kaaaa, kaaaaa enz...
{n,m}	Min én max aantal keer	ka{2,3} matcht alleen maar kaa en kaaa
|	Eén van de alternatieven moet voorkomen	(k|K|c|C)at matcht met kat of Kat of cat of Cat

POSIX	Extended	Betekenis/ASCII	Voorbeeld
[:digit:]	\d	een cijfer [0-9]	
	\D	alles behalve een cijfer [^0-9]	/\D\D\D/ matcht met kat
[:space:]	\s	spatie karakter (ook tab, newline,return) [\t\r\n\v\f]	/\D\D\s\D\D\D/ matcht 'de kat'
	\S	geen spatie karakter [^ \t\r\n\v\f]	/\S\S\s\S\S\S/ matcht 'de kat'
[:alnum:]	\w (zonder _)	Letters en cijfers [a-zA-Z0-9]	
[:alpha:]		Letters [a-zA-Z]	
[:blank:]		Spatie en tab [\t]	
[:lower:]		Lowercase letters [a-z]	
[:print:]		Alle afdrukbare karakters [\x20-\x7E]	
[:punct:]		punctuation karakters [\]\[!"#$%&'()*+,./:;<=>?@\^_`{|}~-]	
[:upper:]		alle upercase letters [A-Z]	
[:xdigit:]		Hexadecimale karakters [0-9A-Fa-f]	
	\b	woordgrens (bv spatie)	/kat\b/ matcht met 'kat ' maar niet met 'katten'
	\B	geen woordgrens	/kat\B/ matcht met 'katten ' maar niet met 'kat '
	\n	newline karakter (ASCII 10)	
	\r	return karakter (ASCII 13)	
	\t	tab karakter	
	\w	een letter, een cijfer én de underscore	
	\W	een enkel niet \w karakter	

Opgelet: bij bash gebruik je dubbele vierkante haken voor de POSIX notatie bv [[:upper:]]

11 OVERZICHT COMMANDO'S

If we spoke a different language, we would perceive a somewhat different world.
-- Ludwig Wittgenstein.

Commando	Verklaring	Voorbeeld gebruik
apropos	Opvragen van alle commando's die met een bepaald keyword te maken hebben	apropos date
basename	Afdrukken van een bestandsnaam zonder het pad	basename */usr/bin/perl*
bc	Tellen (ook achter de komma)	echo 2.1+3.5 | bc
cat	Toont de inhoud van een bestand	cat */etc/passwd*
cut	Substring gebruiken	date | cut -c 12-16
cd	Change directory, Veranderen van directory	cd /usr/share
chmod	Verandert de bestandsrechten	chmod 755 *mijnscript.sh*
chown	Verandert de bestandseigenaar	chown *student1 tekstbestand.txt*
chgrp	Verandert de bestandsgroep	chgrp *studenten tekstbestand.txt*
clear	Scherm leegmaken	clear
cd	Veranderen van directory	cd /home/jancelis
comm	Vergelijken gesorteerde bestanden (byte per byte)	comm -1 -2 *versie1.c versie2.c*
cp	Copieren van bestand(en)	*cp tekstbestand.txt tekstbestand.bak*
diff	Toont verschillen tussen 2 bestanden	diff *versie1.c versie2.c*
dirname	Toont enkel het path	dirname */usr/bin/perl*
df	Disk free, Tonen van vrije schijfruimte	df -h
dos2unix	Omzetten van dos txt formaat naar unix txt formaat	dos2unix *script.bat*
du	Disk usage, Toont de gebruikte schijfruimte van een bepaalde directory	du -s -h */usr/share/doc*
echo	Toont iets op het scherm	echo "Hello world!"
env	Toont alle omgevingsvariabelen	env
expr	Telt of vergelijkt waarden	expr 1 + 2
fdisk	Toont de partitieindeling van de schijven	sudo fdisk -l

find	Zoekt naar een bestand met een bepaalde naam	find ~ -name "*tekstbestand.txt*"
finger	Toont gebruikersinformatie	finger root
free	Toont het gebruikte geheugen	free
g++	Compileren van cpp code	g++ -o *programma source.cpp*
gcc	Compileren van c code	gcc -o *programma source.c*
grep	Zoekt naar string in bestanden	grep -i "paswoord" *
head	Begin van een tekstbestand tonen	head -n 5 */etc/passwd*
id	Gebruikersinformatie afdrukken	id -u kdguntu
kill	Afmaken van een proces	kill -9 1443
last	Toont een geschiedenislijst met de inlogtijd van gebruikers	last
lastlog	Toont de laatste logintijd van elke gebruiker	lastlog
less	Toont inhoud van een bestand/toont een bestand scherm per scherm	ls /usr/bin | less less *tekstbestand.txt*
let	Tellen	som=3+3;echo $som
ln -s	Logische link leggen (shortcut naar een bestand)	ln -s*/etc/passwd /root/paswoordenlink*
locate	Zoekt in de locate db naar de lokatie van bestanden. Deze databank locate db kan je updaten met updatedb	locate http
ls	Toont een lijst met alle bestanden en directories	ls -al
man	Opvragen van de helppagina van een bestand	man kill
mkdir	Aanmaken van een directory	mkdir backup
mkpasswd	Encrypteren van een paswoord	mkpasswd -m md5 paswoord
more	Toont output scherm per scherm	ls /usr/bin | more
mv	Verplaatsen of hernoemen van een bestand	mv *tekstbestand.txt nieuwenaam.txt*
nano	Editeren van een bestand	nano *tekstbestand.txt*
passwd	Aanpassen van het paswoord	passwd kdguntu
pidof	Geeft de PID(s) van een programma	pidof bash
ps	Tonen van een proceslijst	ps -auxww
read	Leest input van een gebruiker of	read x

	bestand (of ook pauze)	
rm	Verwijderen van een bestand	rm -f *tekstbestand.txt*
rm -r	Verwijderen van bestanden en directories	rm -r ./backup/*
sed	stream editor. Editeert bestanden aan de hand van commando's die je aan sed meegeeft.	sed -e 's/$/\r/' unixformaat.txt > dosformaat.txt
sleep	Wacht een aantal seconden (bv 5)	sleep 5
sort	Sorteert een bestand	sort */etc/passwd*
ssh	Openen van een geëncrypteerde telnet verbinding	ssh 192.168.1.41
su	Inloggen als een andere gebruiker	su -
sudo	Commando uitvoeren met root rechten	sudo fdisk -l
tail	Einde van een tekstbestand tonen	tail -n 5 */etc/passwd*
tar	Inpakken/uitpakken van tar bestanden	tar zxvf *programma.tar.gz*
tee	Schrijft de input naar de output of naar een bestand	ls | tee bestand.txt
test	testen van een conditie of expressie	test -f bestaandbestand.txt
touch	Aanmaken of aanraken van een bestand	touch lockfile
tr	Inline vervangen of verwijderen van characters	cat */etc/passwd* | tr ':' ';'
umask	Verandert de default rechten van nieuwe bestanden	umask 022
uname	Toont informatie over je systeem	uname -a
uniq	Verwijdert dubbele lijnen uit een gesorteerd bestand of andere input	uniq *gesorteerdbestand.txt*
updatedb	Update de databank met bestanden deze wordt door locate gebruikt	updatedb
uptime	Toont hoelang de machine al draait, en hoe zwaar deze belast is	uptime
vi	Editeren van een bestand	vi *tekstbestand.txt*
wc	Word count, tellen van woorden, lijnen en characters	echo "Aantal lijnen:" ; wc -l */etc/passwd*

whatis	Opvragen van een korte omschrijving van de functie van een commando	whatis wc
which	Zoeken naar de lokatie van een bestand in het pad	which apache2
who	Toont wie er allemaal ingelogd is op een systeem	who
whoami	Toont met welke username je ingelogd bent	whoami

12 GERAADPLEEGDE WERKEN

A tutorial for moving to autoconf en automake
http://mij.oltrelinux.com/devel/autoconf-automake/
Geraadpleegd op 25/02/2014

Bash Parameter Substitution
http://tldp.org/LDP/abs/html/parameter-substitution.html
Geraadpleegd op 14/02/2013

Bash Internal Variables
http://tldp.org/LDP/abs/html/internalvariables.html
Geraadpleegd op 24/02/2013

Character Classes and Bracket Expressions
http://www.gnu.org/software/grep/manual/html_node/Character-Classes-and-Bracket-Expressions.html
Geraadpleegd op 13/02/2013

GNU make
https://www.gnu.org/software/make/
Geraadpleegd op 25/02/2014

Makefile Tutorial
http://mrbook.org/tutorials/make
Geraadpleegd op 26/02/2014

Manipulating Strings
http://tldp.org/LDP/abs/html/string-manipulation.html
Geraadpleegd op 31/01/2013

Unix regex(7)
Op te vragen met het commando: *man 7 regex*

The conditional expression
http://wiki.bash-hackers.org/syntax/ccmd/conditional_expression
Geraadpleegd op 31/01/2013

Writing Make Files
http://www.cs.bu.edu/teaching/cpp/writing-makefiles/
Geraadpleegd op 11/03/2014