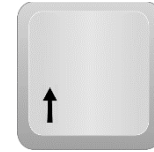

Inleiding scripting

Aanraders

- Vorige commando's: Pijl naar boven



- Command completion

of bij meerdere mogelijkheden



- `sudo apt-get install vim`

- ◆ met syntax highlighting, pijltjes die werken

Shell script

■ Uitvoeren shell script:

- ◆ Eerste lijn is `#!/bin/bash`
- ◆ Bestand is executable: `chmod +x bestand.sh`
- ◆ Starten met `./bestand.sh`

Basics

■ Pipes en redirection

- ◆ input, output en error kanaal
 - STDIN, STDOUT, STDERR (0, 1, 2 resp.)
 - echo "hello" > tekst.txt #STDOUT naar bestand
 - mkdir test 2> error.txt #STDERR naar bestand
 - mkdir test 2> /dev/null #STDERR weg
- ◆ STDOUT commando *ps* naar STDIN *grep*
 - ps -ef | grep apache
- ◆ STDIN naar cut STDOUT naar sort
 - cut -d: -f7 </etc/passwd | sort

Basics

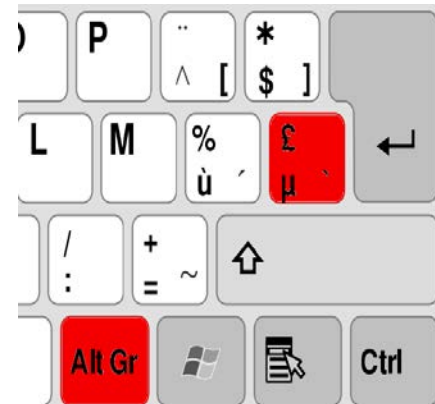
■ Variabelen

- ◆ `mijnvar="hello"`
- ◆ `echo $mijnvar`
- ◆ `echo "$mijnvarworld"` `# werkt niet`
- ◆ `echo "${mijnvar}world"` `# werkt wel`

Backquotes

■ Backquotes voeren het commando eerst uit en vullen de output in

- ◆ ALTGR- μ (meestal 2 keer)
- ◆ Tonen aantal lijnen van bestand /etc/passwd:
 - `wc -l /etc/passwd`
44 /etc/passwd
 - `cat /etc/passwd | wc -l`
44
- ◆ Gebruik van de output:
 - `echo "Er zijn `cat /etc/passwd | wc -l` users"`



Backquotes

- ...
- Als je de Backquotes moeilijk te typen of onduidelijk vindt...
- \$() gaat ook
-

```
peter@PC1:~$ echo "Er zijn `cat /etc/passwd | wc -l` users"
Er zijn 53 users
peter@PC1:~$ echo "Er zijn $(cat /etc/passwd | wc -l) users"
Er zijn 53 users
```

Basics

■ Commando's

- ◆ echo: textoutput tonen op scherm
- ◆ cat: tonen inhoud van een bestand
- ◆ cut: knippen, ook met velden en separators
- ◆ sort: sorteren bestand
- ◆ uniq: tonen dubbele of niet dubbele lijnen
- ◆ wc: tellen van lijnen, woorden, karakters
- ◆ head: eerste n lijnen tonen
- ◆ tail: laatste n lijnen tonen
- ◆ grep: zoeken op patronen in bestand (ook regex)
- ◆ find: zoeken naar bepaalde bestanden

Bash parameters

- \$0 bestandsnaam
- \$1, \$2, \$3,... 1ste, 2de, 3de argument

Output van een commando in een variabele

```
#!/bin/bash
```

```
# Functie: Toont versie van Ubuntu
```

```
versie=`lsb_release -r | cut -f2`
```

```
echo "Dit toestel draait Ubuntu versie ${versie}"
```

OUTPUT:

```
Dit toestel draait Ubuntu versie 14.04
```

Output van een commando in een variabele

```
#!/bin/bash
# Functie: Toont versie van Ubuntu
versie=`lsb_release -r | cut -f2`
echo "Dit toestel draait Ubuntu versie ${versie}"
```

OUTPUT:

Dit toestel draait Ubuntu versie 14.04

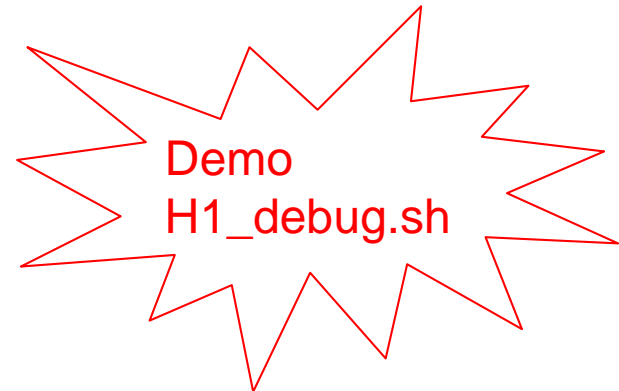
```
-d, --delimiter=DELIM
        use DELIM instead of TAB for field delimiter
```

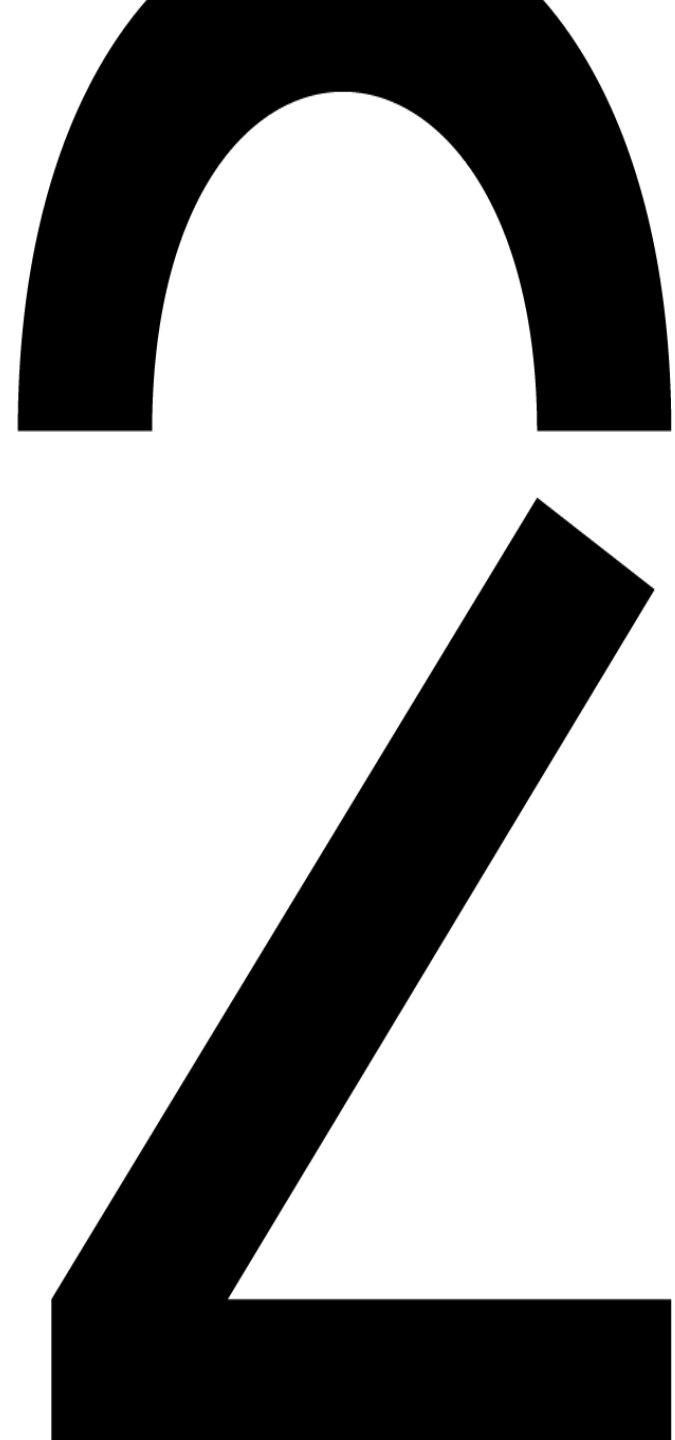
Script debug

- Optie -x gebruiken bij opstarten:
jancelis@kdguntu\$ bash -x ./debug.sh
- Of in de eerste regel van het script de optie -x toe te voegen

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

```
peter@PC1:~/CS2$ ./H1_debug.sh  
+ var=1  
+ echo 3  
3  
peter@PC1:~/CS2$
```





Herhaling loops

Lees Hoofdstuk 1

IF

WHILE

FOR

Bash Flow Control -if

■ if

```
#!/bin/bash
```

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

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

Note:

```
a=1 ; b=2 ; if [ $a -eq $b ] ; then echo "equal" ; else echo "not equal" ; fi
```

Bash Flow Control -if test wordt []

■ if

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

■ Goede regel: ALTIJD spatie voor én na []

IF
WHILE
FOR

Bash: while

■ while

```
#!/bin/sh
```

```
teller=0
```

```
while [ $teller -lt 10 ]
```

```
do
```

```
echo -n "$teller "
```

```
let teller+=1          # eentje bijtellen
```

```
done
```

```
echo -e "\nEinde"
```

Bash operators uit "man test"

■ vergelijken

-	string	numeriek
gelijk	<code>x = y</code>	<code>x -eq y</code>
niet gelijk	<code>x != y</code>	<code>x -ne y</code>
groter	<code>x > y</code>	<code>x -gt y</code>
kleiner	<code>x < y</code>	<code>x -lt y</code>
groter/gelijk	<code>x >= y</code>	<code>x -ge y</code>
niet leeg	<code>-n x</code>	
leeg(zero)	<code>-z x</code>	

■ bestanden nakijken

- `-d` is directory
- `-f` is file
- `-e` bestand bestaat (exists)
- `-r` leesrechten op bestand
- `-s` bestaat en niet leeg
- `-w` schrijfrechten op bestand
- `-x` exe rechten op bestand

IF
WHILE
FOR

Bash for

■ commando output met for

```
#!/bin/bash
```

```
# Functie: lijst tonen van bestanden in een  
directory
```

```
directory="/usr/share/backgrounds"
```

```
for bestand in `ls $directory`
```

```
do
```

```
    echo "Bestand: $bestand"
```

```
done
```

Bash for

■ Uitlezen van een bestand

```
#!/bin/bash
```

```
# Functie: Lijn per lijn het passwd  
bestand tonen
```

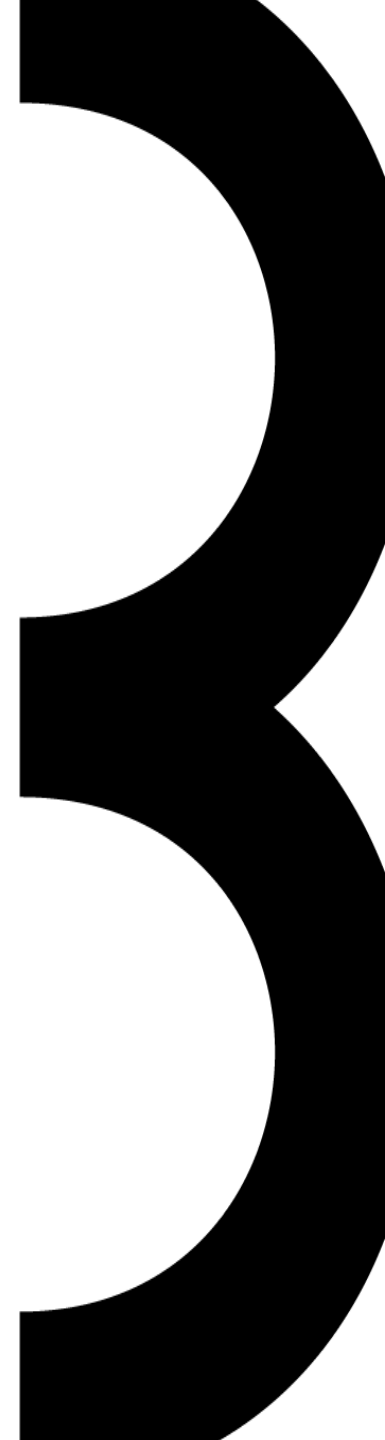
```
bestand="/etc/passwd"
```

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

Bash for & IFS

- Extra bij for-lus
 - The Bash for loop splits using a whitespace (space, tab or newline).
 - Dit noemen we de IFS parameters:
 - `Internal field separator`
- Wil jij dus dat hij enkel bij een newline split dan moet je de IFS aanpassen naar enkel newline:
 - Opzetten bij het begin van je script
 - `IFS=$'\n'`
 - Afzetten op het eind van je script
 - `unset IFS`

Bash Best Practices



Overzichtje

- **Eerste lijn**
- **Help**
- **Inputvalidatie**
- **Exit code**
- **Variabelen**
- **Commentaar**
- **Root-rechten**
- **Fouten naar STDERR**
- **Fouten bevatten nodige info**

Basis script

- **De eerste lijn van het script specificeert de soort shell of het programma waarmee het draait**

```
script="script.sh"
```

```
echo '#!/bin/bash' > ${script}
```

```
echo "echo \"Hello\"" >> ${script}
```

```
chmod +x ${script}
```

- ◆ Bij perl #!/usr/bin/perl
- ◆ Bij python #!/usr/bin/python

Argumenten en --help voorzien

```
#!/bin/bash
# Functie: drukt het eerste argument af
if [ $# -lt 1 -o "$1" = "--help" ] ; then
echo "Usage: `basename $0` arg1 "
echo "Example: `basename $0` Hallo "
    exit 1
fi
echo "Eerste argument is $1"
```

Input validatie

- #!/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

Input validatie

- #!/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

Geef een juiste exit code

- Exit code 0 als alles ok is, niet 0 bij falen

```
#!/bin/bash
```

```
# Functie: Aanmaken directory
```

```
directory="test"
```

```
mkdir "$directory"
```

```
if [ $? != 0 ] ; then
```

```
    echo "$directory aanmaken niet gelukt"  
    >&2
```

```
    exit 1
```

```
else
```

```
    echo "$directory aanmaken gelukt"
```

```
    exit 0
```

```
fi
```

Volg afspraken voor namen van variabelen

- Alle omgevingsvariabelen zijn in
HOOFDLETTERS
 - ◆ Deze zie je met het commando "env"
- Al je eigen, niet geexporteerde variabelen zijn
in kleine_letters

Functie: toont hoeveel plaats je \$HOME gebruikt

warning_du="Uw homedir \$HOME is bijna vol"

schijfruimte_home=`du \$HOME | cut -f1`

echo -n "Homedir \$HOME van user \$USER "

echo "bevat \$schijfruimte_home bytes"

if [\$schijfruimte_home -lt 100000]; then

echo "\$warning_du"

fi

Zet voldoende in commentaar

```
#!/bin/bash
```

```
# Functie:      Zoekt naar grote bestanden  
#              In een bepaalde directory
```

```
# Arguments:    Arg1 is een directory
```

```
# Auteur:       jan.celis@kdg.be
```

```
# Copyright:    2014 GNU v3 jan.celis@kdg.be
```

```
# Versie:       0.1
```

```
# Requires:     Standaard shell find commando
```

```
•
```

```
if [ -d "$1" ]; then
```

```
    find $1 -type f -size +100M
```

```
fi
```


Gebruik sudo indien nodig

```
#!/bin/bash
if [ "$(id -u)" != "0" ]; then
    echo "Dit script moet je als root uitvoeren"
    1>&2
    exit 1
fi
sudo cat /etc/shadow
```

Fouten stuur je naar STDERR

■ Kanalen STDIN, STDOUT en **STDERR**

- ◆ nummers 0, 1 en 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
```

Fouten bevatten een programma en functienaam

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

Fouten bevatten een programma en functienaam

- #!/bin/bash
- # Functie: Geeft een foutbericht bij volle schijf
- err_df="Fout in \$0:Je schijf is vol"
- func_df()
- {
df_percent=`df /|tr -s ' ' |cut -d' ' -f5|tail -n1`

```
peter@PC1:~$ df /
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/sda1        9711136 5429124   3765660   60% /
peter@PC1:~$ df / | tr -s ' '
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/sda1 9711136 5428956 3765828 60% /
peter@PC1:~$ df / | tr -s ' ' | cut -d ' ' -f 5
Use%
60%
peter@PC1:~$ df / | tr -s ' ' | cut -d ' ' -f 5 | tail -n1
60%
peter@PC1:~$ □
```

Dependencies

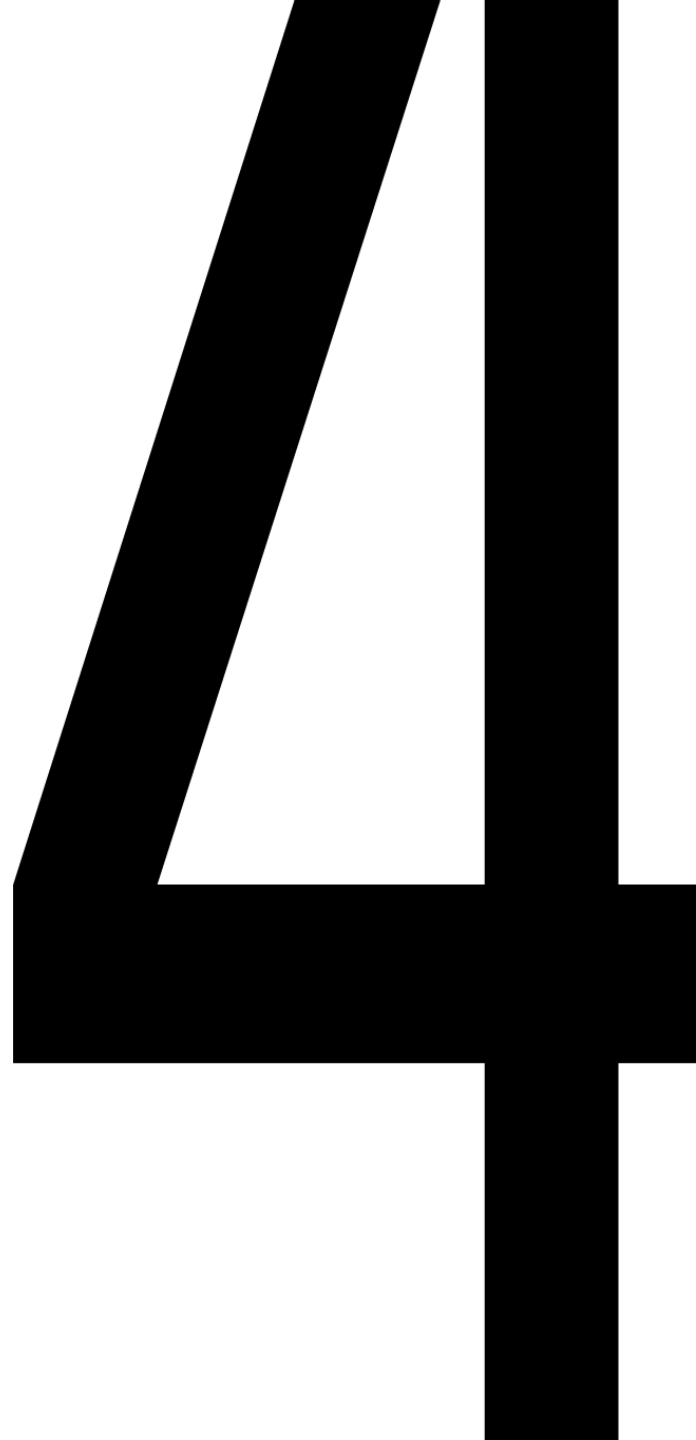
- Nakijken of een commando bestaat voor je het gebruikt
 - ◆ `error_geen_ab="Het programma ab is niet geïnstalleerd"`
`command -v ab >/dev/null || (echo $error_geen_ab >&2 && exit 1);`
 - ◆ `which ab >/dev/null || (echo $error_geen_ab >&2 && exit 1);`
- Nakijken of een programma fouten gaf of niet
`error_url= "de url is niet bereikbaar"`
`curl -o /dev/null --silent --head --connect-timeout 1`
`${url}`
`if [$? -ne 0]; then`
`echo ${error_url} && exit 1`
`fi`

Dependencies

- Alle andere programma's en bibliotheken die het script nodig heeft staan aangegeven in commentaar
- Bij welk pakket hoort een commando?

```
jancelis@kdguntu:~$ which ab
/usr/bin/ab
jancelis@kdguntu:~$ dpkg -S
/usr/bin/ab
apache2-utils: /usr/bin/ab
```

Bash Functies en Tellen



Functies

Tellen

Functies

Tellen

Bash functie

```
#!/bin/bash
```

```
# Functie: Declaratie van de function "functie"
```

```
#           Oproepen van functie
```

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

```
functie
```

Bash functie eigenschappen

- ❑ Een functie MOET gedefinieerd zijn VOOR je ze gebruikt
- ❑ Een functie mag NIET leeg zijn
 - ▣ ook niet enkel commentaar
- ❑ De argumenten \$1, \$2,... zijn NIET de argumenten van het script maar van de functie

Bash functie met argument

```
#!/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"
```

Voorbeeld:
H4_kleur.sh

Bash functie met argument

- Wat extra info:
- The ANSI [\[1\]](#) escape sequences set screen attributes, such as bold text, and color of foreground and background.
- [DOS batch files](#) commonly used ANSI escape codes for *color* output, and so can Bash scripts.
- <http://tldp.org/LDP/abs/html/colorizing.html>
- http://misc.flogisoft.com/bash/tip_colors_and_formatting

Bash functions: return waarde

- Een functie kan in bash eigenlijk alleen maar **gelukt** of **niet gelukt** teruggeven

```
function checkexist(){  
    if [ -f "$1" ]; then  
        return 0  
    else  
        return 1  
    fi  
}
```

```
if checkexist "/etc/passwd" ; then echo passwd ok; fi
```

Bash functions: return

```
#!/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 "ok")  
# of resultaat=`tooninkleur "ok"`  
echo $resultaat
```

Functies

Tellen

Bash tellen

- `+` `-` `/` `*` `%` `**` (plus, min, delen, maal, mod, expon.)

- Met `expr`

- `a=1`
`b=2`
`som=`expr $a + $b``

- Met `let`

- `let "som=$a+$b"`

- In `bash`

- `som=$(($a + $b))`
`echo $som`

- ◆ **Opm: `echo $((3/2))` → geeft 1**
 - ◆ **Voor komma getallen gebruik je `"bc"`**

Voorbeeld:
`H4_obase_ibase.sh`

Bash tellen

Voorbeeld:
H4_obase_ibase.sh

BC

Simpel script:

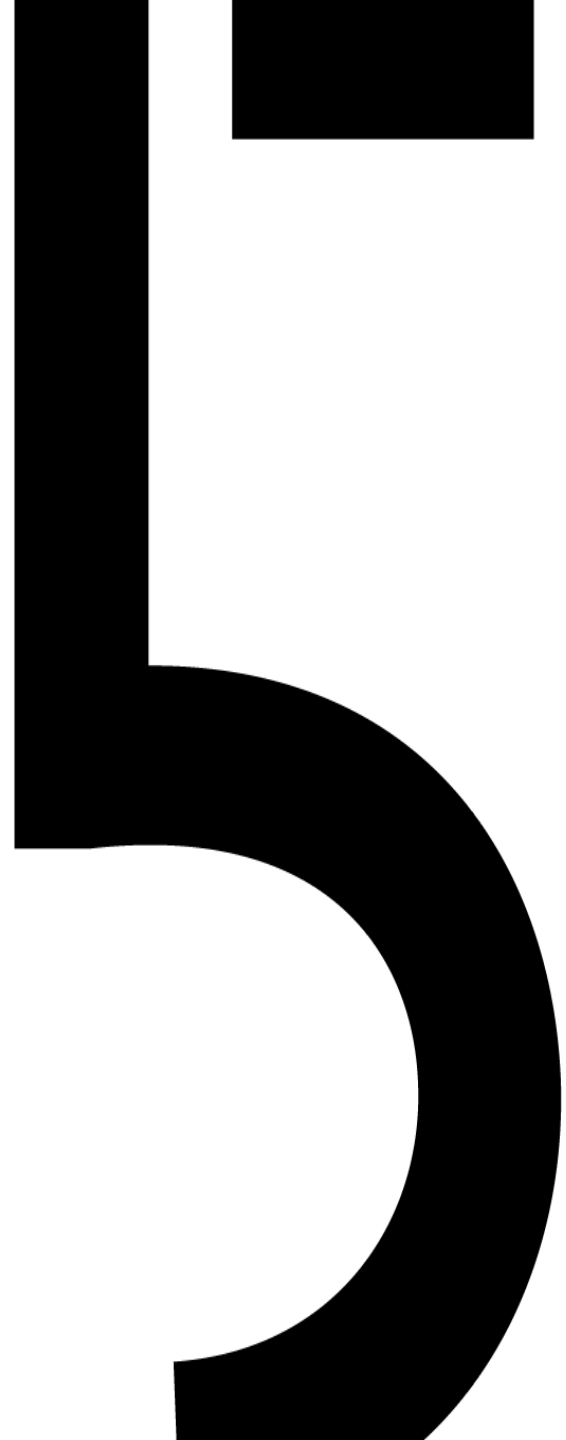
- `#!/bin/bash`
- `echo '6.5 / 2.7' | bc:`

Je kan bc parameters zetten:

- **Scale** = hoeveel getallen na komma
 - geen afronding (evt. Zelf berekenen → functie schrijven...)
- Conversies tussen talstelsels:
 - **Obase** = basis voor output
 - **Ibase** = basis voor input

Voorbeeld:
./H4_bc_roundup.sh

Bash Parameter Substitutie



PARAMETER SUBSTITUTION

- interne vars
- vars
- replace
- substring
- trim
- length
- upper/lower

Voorbeeld interne variabelen

Slides_H5_MoederKind.sh

```
#!/bin/bash
```

```
echo Dit moeder script heet $0
```

```
echo Argument 1 is $1
```

```
# eindeloos kind 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 echo van de lijn hiervoor
```

```
chmod +x child.sh ; ./child.sh & # Opstarten in achtergrond
```

```
sleep 2 # moeder slaapt 2 seconden
```

```
echo kind nr $! wordt door moeder afgemaakt
```

```
kill $! # PID laatste achtergrondproces (child.sh)
```

```
echo moeder $0 nr $$ pleegt zelfmoord
```

```
kill $$
```

Gebruik van IFS en \$@

Slides_H5_test_ifs.sh
Slides_H5_test_ifs_noset.sh

```
#!/bin/bash
```

```
# Args:  Arg $1 is een bestand
```

```
#          OF via stdin
```

```
IFS=$'\n'
```

```
for lijn in $(cat "$@")
```

```
do
```

```
    echo $lijn
```

```
done
```

```
unset IFS
```

Oproepen met: `cat /etc/passwd | ./script.sh`
 of
 `./script.sh /etc/passwd`

Variabele `${var}`

- In plaats van `$var` gebruiken we `${var}`
var="hallo"
echo "\$varmetdezoo" #werkt niet
echo "\${var}metdezoo" #werkt wel!

Default waarde variabele \${var:-default}

```
#!/bin/bash
```

```
# Functie: Zoeken naar grote bestanden,  
default >MB
```

```
# Args: Arg $1 MAG meegegeven worden
```

```
# Als $1 leeg is, krijgt het een default waarde
```

```
default_waarde="10"
```

```
size=${1:-$default_waarde} # of =${1:-10}
```

```
find . -iname "*" -size "+${size}M"
```


Lege variabelen geven error

`${var?error_message}`

■ Een variabele die leeg is geeft:

- ◆ foutbericht
- ◆ eindigt met exit 1

```
#!/bin/bash
```

```
# Functie: Grote bestanden vinden
```

```
# Args: Arg $1 MOET meegegeven worden
```

```
# $1 leeg => einde met exit code 1
```

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

```
# Script komt enkel hier wanneer $1 werd  
ingevuld
```

```
find . -iname "*" -size "+${size}M"
```

Replace

`${var/zoekterm/vervangterm}`

```
#!/bin/bash
```

```
# Functie: Replace in een string
```

```
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}
```

Substring \${var:positie:lengte}

H5_substring.sh

```
#!/bin/bash
```

```
url="http://www.kdg.be/index.html"
```

```
echo String ${url}
```

```
echo Eerste 7 karakters knippen ${url:7}
```

```
# Haakjes of spatie is escape van positie
```

```
echo Laatste 4 karakters knippen ${url: -4}
```

```
# of echo Laatste 4 karakters ${url:(-4)}
```

```
echo Eerste 4 karakters weergeven ${url:0:4}
```

```
echo Karakter 8 tot 18 weergeven ${url:7:10}
```

trim

Slides_H5_Greedy.sh

```
#!/bin/bash
```

```
# Functie: Verwijder gedeelte string voor/achteraan  
#          greedy en ungreedy
```

```
url="http://www.kdg.be/index.html"
```

```
# Verwijder korste substring http:// vooraan:  
${url#http://}
```

```
# Verwijder langste substring http*. vooraan:  
${url##http*.}
```

```
# Verw. korste substring non-greedy .* achteraan:  
${url%.*}
```

```
# Verw. langste substring greedy .* achteraan:  
${url%%.*}
```

Lengte string

```
#!/bin/bash
```

```
url="http://www.kdg.be/index.html"
```

```
echo Lengte van de string: ${#url}
```

Uppercase/lowercase

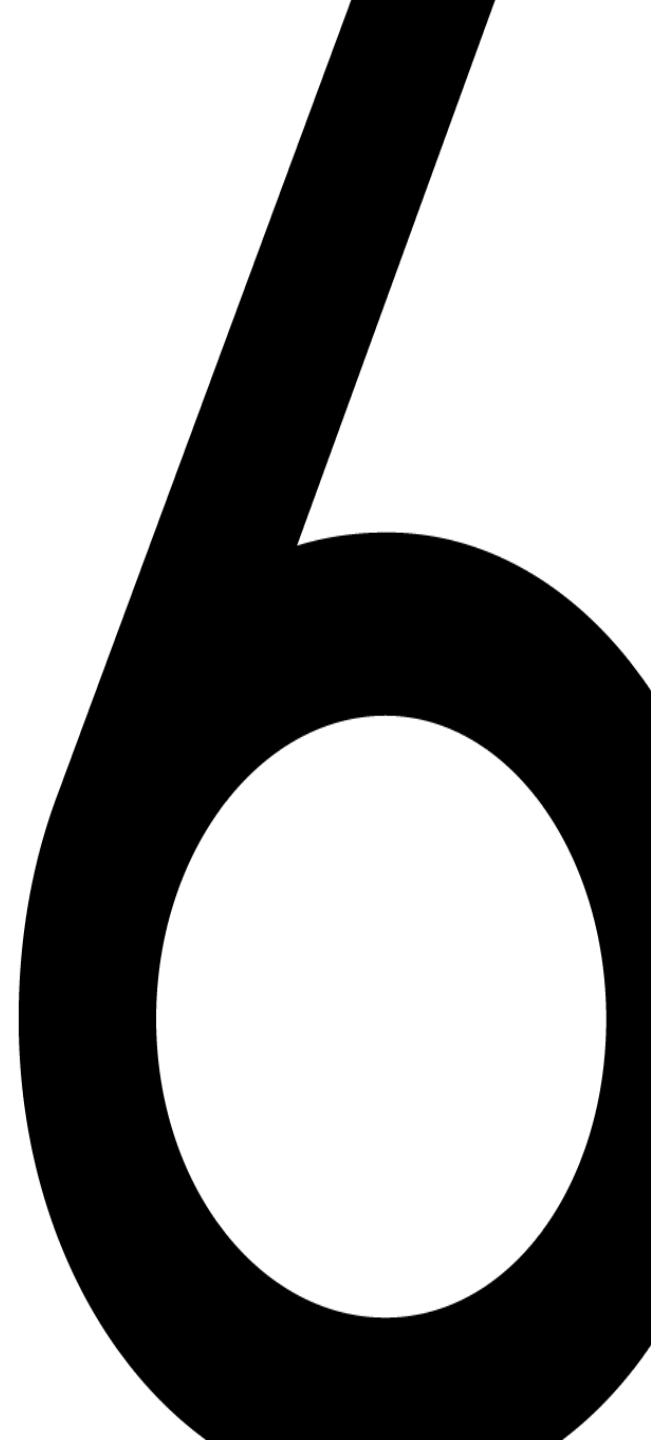
```
#!/bin/bash
```

```
url="http://www.kdg.be/index.html"
```

```
echo Alles uppercase: ${url^^}
```

```
echo Alles lowercase: ${url,,}
```

Bash Regex



REGEX

- soorten
- expressies
- =~
- groups
- extglob

Soorten regex

■ POSIX2.0 Regex

- ◆ Extended Regex (ERE)
 - `egrep/bash`
- ◆ Oud: Basic Regex(BRE)
 - escapen van ? + { | (en)
 - standaard grep (zie CS1...)

■ Perl Compatible Regex (PCRE)

- UTF-8 en UNICODE support
- perl/python/java/javascript/c#
- grep -P

Extended en basic regex voorbeeld

■ Extended regex met grep (egrep):

- ◆ `grep -E "groen|rood" bestand.txt`

■ Basic regex met grep

- ◆ `grep "groen\\|rood" bestand.txt`

Enkele regex expressies

■ .

■ [a-z]

■ [0-9]

■ [a|b]

■ a{4}

■ a{1,4}

■ +

■ *

Enkele regex expressies

- . 1 karakter
- [a-z] karakter a tot en met z in kleine letters
- [0-9] getal 0 tot en met 9
- [a|b] karakter a of karakter b
- a{4} vier keer a
- a{1,4} van één tot 4 keer a
- + 1 tot n keer
- * 0 tot n keer

POSIX: Soorten karakters extended regex

- `[[:digit:]]` een cijfer
- `[[:space:]]` spatie, tab, newl, return
- `[[:alnum:]]` letters en cijfers
- `[[:alpha:]]` letters
- `[[:blank:]]` spatie en tab
- `[[:lower:]]` lowercase letters
- `[[:print:]]` afdrukbare karakters

= ~ operator

```
#!/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
```

OPGELET: \$regex is NOOIT met dubbele quotes!

`${BASH_REMATCH[0]}`

- `${BASH_REMATCH[0]}` is een array met het resultaat van de match. Element 0 is de hele match

```
#!/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

regex voorbeeld: email adres

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

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

regex groups

- Een group selecteert met haakjes een onderdeel van de regex
- Het resultaat zit in array `n` `${BASH_REMATCH[n]}`

```
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

"Regex" met parameter substitution

```
#!/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
```

Linux

- Vraag:
 - Ik begrijp de opbouw van de regex niet helemaal die gebruikt wordt in de parametersubstitutie in volgende voorbeelden:
 - 1) zie cursus p 42:
`content=${content##*+([[:digit:]])}` Dit verwijdert alles tot en met het eerste cijfer. Wat betekent de combinatie van * en + hier?
 - 2) `content=${content##*([[:space:]])}` Dit verwijdert alle leading spaties. Ook hier begrijp ik niet goed wat het sterretje wil zeggen.

Linux

- Verklaring:
 - Extglob = extended globbing
 - Zoals regular expressions, maar met een paar eigenaardigheden
 - quantifiers (+ * ? ...) hebben dezelfde betekenis als in regex, maar moeten voor het patroon (dat tussen ronde haakjes staat) staan ipv erachter

Linux

- Verklaring:
 - <http://wiki.bash-hackers.org/syntax/pattern>
 - Let op: regular expressions belangrijker! (Slides als randinfo.)

Linux

- Verklaring:

Normal pattern language

Sequence	Description
<code>*</code>	Matches any string , including the null string (empty string)
<code>?</code>	Matches any single character
<code>x</code>	Matches the character <code>x</code> which can be any character that has no special meaning
<code>\x</code>	Matches the character <code>x</code> , where the character's special meaning is stripped by the backslash
<code>\\</code>	Matches a backslash
<code>[...]</code>	Defines a pattern bracket expression (see below). Matches any of the enclosed characters at this position.

Linux

- Verklaring:

Bracket expressions



The bracket expression `[...]` mentioned above has some useful applications:

Bracket expression	Description
<code>[XYZ]</code>	The "normal" bracket expression, matching either <code>X</code> , <code>Y</code> or <code>Z</code>
<code>[X-Z]</code>	A range expression: Matching all the characters from <code>X</code> to <code>Y</code> (your current locale , defines how the characters are sorted!)
<code>[:class:]</code>	Matches all the characters defined by a POSIX® character class: <code>alnum</code> , <code>alpha</code> , <code>ascii</code> , <code>blank</code> , <code>cntrl</code> , <code>digit</code> , <code>graph</code> , <code>lower</code> , <code>print</code> , <code>punct</code> , <code>space</code> , <code>upper</code> , <code>word</code> and <code>xdigit</code>
<code>[^...]</code>	A negating expression: It matches all the characters that are not in the bracket expression
<code>[!...]</code>	Equivalent to <code>[^...]</code>
<code>[...] or [-...]</code>	Used to include the characters <code>]</code> and <code>-</code> into the set, they need to be the first characters after the opening bracket
<code>[=C=]</code>	Matches any character that is equivalent to the collation weight of <code>C</code> (current locale!)
<code>[.SYMBOL.]</code>	Matches the collating symbol <code>SYMBOL</code>

Linux

- Verklaring:

Examples

Some simple examples using normal pattern matching:

- Pattern `"Hello world"` matches
 - `Hello world`
- Pattern `[Hh]"ello world"` matches
 - `⇒ Hello world`
 - `⇒ hello world`
- Pattern `Hello*` matches (for example)
 - `⇒ Hello world`
 - `⇒ Helloworld`
 - `⇒ HelloWoRlD`
 - `⇒ Hello`
- Pattern `Hello world[:punct:]` matches (for example)
 - `⇒ Hello world!`
 - `⇒ Hello world.`
 - `⇒ Hello world+`
 - `⇒ Hello world?`
- Pattern `[[:backslash:]]Hello[[:vertical-line:]]world[[:exclamation-mark:]]` matches (using collation sybols)
 - `⇒ \Hello|world!`

Linux

- Verklaring:

Extended pattern language

If you set the [shell option](#) `extglob`, Bash understands some powerful patterns. A `<PATTERN-LIST>` is one or more patterns, separated by the pipe-symbol (`PATTERN|PATTERN`).

<code>?(<PATTERN-LIST>)</code>	Matches zero or one occurrence of the given patterns
<code>*(<PATTERN-LIST>)</code>	Matches zero or more occurrences of the given patterns
<code>+(<PATTERN-LIST>)</code>	Matches one or more occurrences of the given patterns
<code>@(<PATTERN-LIST>)</code>	Matches one of the given patterns
<code>!(<PATTERN-LIST>)</code>	Matches anything except one of the given patterns

Linux

- Verklaring:
 - Extglob = extended globbing
 - Zoals regular expressions, maar met een paar eigenaardigheden
 - quantifiers (+ * ? ...) hebben dezelfde betekenis als in regex, maar moeten voor het patroon (dat tussen ronde haakjes staat) staan ipv erachter

Linux

- Vraag:
 - 1) `content=${content##* + ([:digit:])}`
 - 2) `content=${content##* ([:space:])}`

Linux

- Toepassing:
 - `content=${content##*+([[:digit:]])}`
 - String functie om te knippen
 - `#` een zo klein mogelijk string vinden
 - `##` een zo groot mogelijk string vinden
 - 1 of meer cijfers (equivalent met `[[:digit:]]+` in regex)
 - `*` = pattern = any string (including NULL)
 - Besluit:
 - Zoeken de grootst mogelijke string die start met eender wat en eindigt op 1 of meer getallen en knippen die weg.

Linux

- Toepassing:
 - `content=${content##*+([[:digit:]])}`
 - String functie om vooraan te knippen
 - `#` een zo klein mogelijk string vinden
 - `##` een zo groot mogelijk string vinden
 - 1 of meer cijfers (equivalent met `[[:digit:]]+` in regex)
 - `*` = pattern = any string (including NULL)
 - Besluit:
 - Zoeken de grootst mogelijke string die start met eender wat en eindigt op 1 of meer getallen en knippen die weg.

Linux

- Toepassing:
 - `content=${content##*([[:space:]])}`
 - String functie om vooraan te knippen
 - `#` een zo klein mogelijk string vinden
 - `##` een zo groot mogelijk string vinden
 - 0 of meer cijfers (zie extended patterns)
 - Besluit:
 - Zoeken naar 0 of meer spaties en deze wegnippen.

Hulpmiddelen

- <http://regexraptor.net/>
- <http://www.regexr.com/>
- CheatSheet in cursus...



Computersystemen 2 Labo

**AutoConf
&
Automake**

Autoconf & Automake

- User gets application
- User types:
 - ./configure
 - make
 - make install

Structuur

- Structuur:
 - Src
 - Doc
 - Man
 - Scripts
 - Examples

Autoconf & Automake

- sources in src/
- documentation in doc/
- man pages in man/
- scripts in scripts/
 - Things to be installed but not compiled
- examples in examples/

Autoconf & Automake

- sources in src/
- documentation in doc/

• mar

• scri|

– Th

• exa

```
peter@PC1:~/CS2/H7-test2$ ls -R
.:
doc  examples  man  scripts  src

./doc:

./examples:

./man:
helloworld.1

./scripts:

./src:
helloworld.c
```

Autoconf & Automake

- sources in src/
- documentation in doc/

• mar

• scri

– Th

• exa

```
peter@PC1:~/CS2/H7-test2$ ls -R
.:
doc  examples  man  scripts  src
```

```
./doc:
```

```
./examples:
```

```
./man:
```

```
helloworld.1
```

```
./scripts:
```

```
./src:
```

```
helloworld.c
```

```
peter@PC1: ~/CS2/H7-test2/src
#include <stdio.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Autoconf & Automake

- Autoscan

–autoscan tries to produce a suitable configure.ac file (autoconf's driver) by performing simple analyses on the files in the package. This is enough for the moment (many people are just happy with it as permanent). Autoscan actually produces a configure.scan file.

```
peter@PC1:~/CS2/H7-test2$ autoscan
peter@PC1:~/CS2/H7-test2$ ls
autoscan.log  configure.scan  doc  examples  man  scripts  src
peter@PC1:~/CS2/H7-test2$
```

Autoconf & Automake

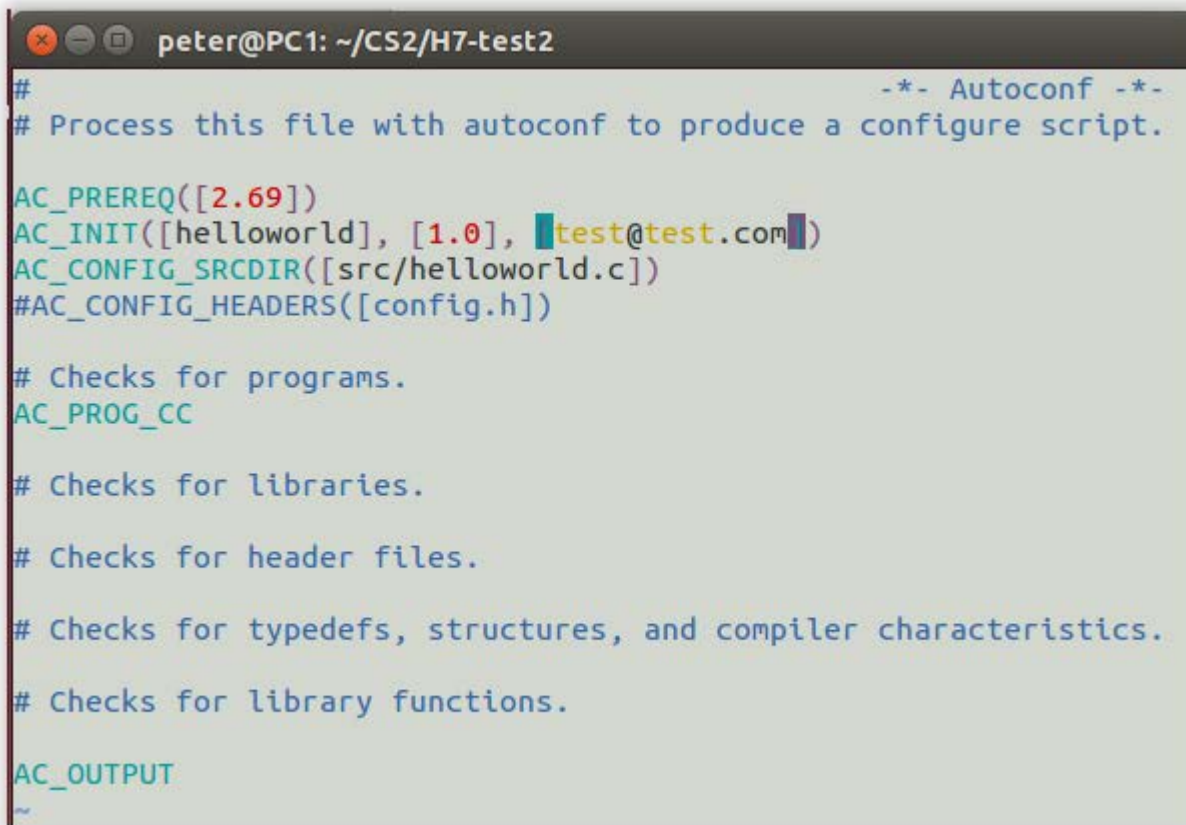
- Autoscan

- Autoscan actually produces a configure.scan file, so we rename manually to configure.ac = file needed by autoconf.

```
peter@PC1:~/CS2/H7-test2$ mv configure.scan configure.ac
peter@PC1:~/CS2/H7-test2$ ls
autoscan.log  configure.ac  doc  examples  man  scripts  src
peter@PC1:~/CS2/H7-test2$
```

Autoconf & Automake

- Manually adapt configure.ac
 - Change AC_INIT to reflect our program.



```
peter@PC1: ~/CS2/H7-test2
# -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.

AC_PREREQ([2.69])
AC_INIT([helloworld], [1.0], [test@test.com])
AC_CONFIG_SRCDIR([src/helloworld.c])
#AC_CONFIG_HEADERS([config.h])

# Checks for programs.
AC_PROG_CC

# Checks for libraries.

# Checks for header files.

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.

AC_OUTPUT
```


Autoconf & Automake

- Let autoconf produce first configure script:

```
peter@PC1:~/CS2/H7-test2$ autoconf
peter@PC1:~/CS2/H7-test2$ ls
autom4te.cache  configure  doc  man  src
autoscan.log    configure.ac  examples  scripts
```

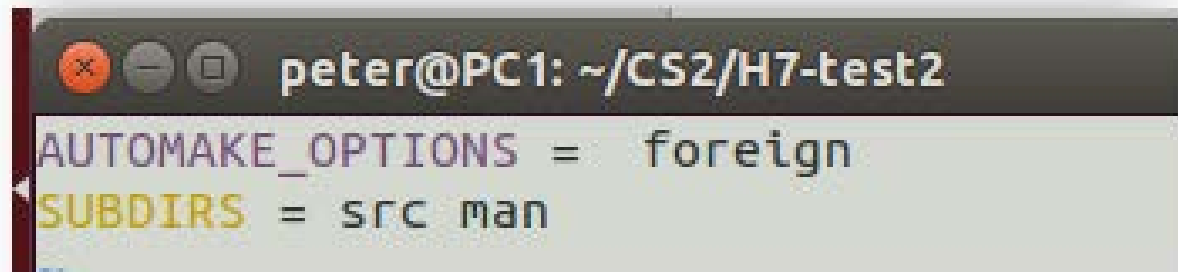
This produces two files: autom4te.cache and configure.

- The first one is a directory used for speeding up the job of autoshell tools, and may be removed when releasing the package.
- The latter is the shell script called by final users. In this status, what the configure script does is just **checking for requirements** as suggested by autoscan, so nothing very conclusive yet.

Autoconf & Automake

- Generate Makefile:

- Vim Makefile.am

A screenshot of a terminal window with a dark background. The title bar shows the window name 'peter@PC1: ~/CS2/H7-test2'. The terminal content shows two lines of text: 'AUTOMAKE_OPTIONS = foreign' and 'SUBDIRS = src man'. The text is color-coded: 'AUTOMAKE_OPTIONS' is purple, 'SUBDIRS' is yellow, and 'foreign' and 'src man' are white.

```
peter@PC1: ~/CS2/H7-test2
AUTOMAKE_OPTIONS = foreign
SUBDIRS = src man
```

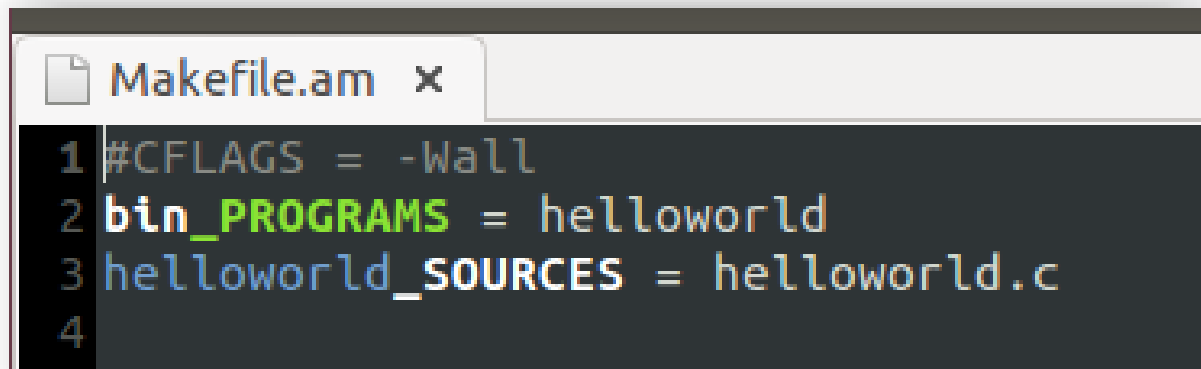
the first line sets the mode automake will behave like. "foreign" means not GNU, and is common for avoiding boring messages about files organized differently from what gnu expects.

The second line shows a list of subdirectories to descend for further work. The first one has stuff to compile, while the rest just needs installing, but we don't care in this file. We now prepare the Makefile.am file for each of these directories. Automake will step into each of them and produce the corresponding Makefile.in file. Those .in files will be used by autoconf scripts to produce the final Makefiles.

Autoconf & Automake

- Generate Makefile:

- Vim src/Makefile.am



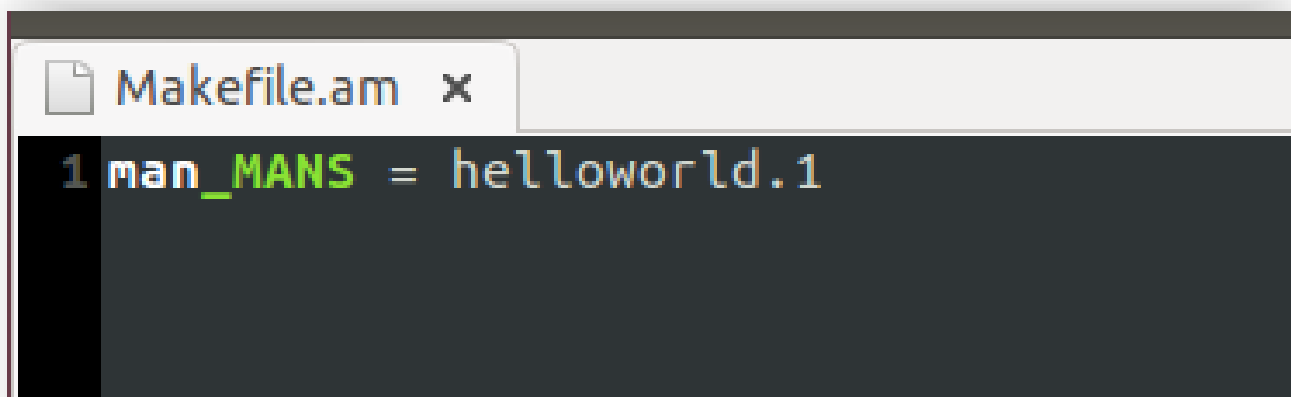
A screenshot of a Vim editor window. The title bar at the top shows a file icon, the name 'Makefile.am', and a close button 'x'. The editor area has a dark background with light-colored text. The text is as follows:

```
1 #CFLAGS = -Wall
2 bin_PROGRAMS = helloworld
3 helloworld_SOURCES = helloworld.c
4
```

Autoconf & Automake

- Generate Makefile:

- Vim man/Makefile.am



A screenshot of a Vim editor window. The title bar at the top shows a file icon, the name 'Makefile.am', and a close button 'x'. The editor area has a dark background and displays the text '1 man_MANS = helloworld.1' on the first line. The line number '1' is in white, 'man_MANS' is in green, and '= helloworld.1' is in white.

Autoconf & Automake

- Generate Makefile:
 - Other makefiles can be generated...
 - Scripts
 - Doc
 - Examples

Autoconf & Automake

- Inform autoconf about the Makefile files
= edit configure.ac
- AM_INIT_AUTOMAKE
 - To initialize automake
- AC_OUTPUT
 - To output makefiles for all the directories above

Autoconf & Automake

- If you are using Autoconf and Automake, you can use the following code to generate a configure script.
 - A snippet of the configure.ac file:
- ```
-*- Autoconf -*-
Process this file with autoconf to produce a configure script.

AC_PREREQ([2.69])
AC_INIT([helloworld], [1.0], [test@test.com])
AM_INIT_AUTOMAKE(helloworld, 1.0)
AC_OUTPUT(Makefile src/Makefile man/Makefile)
AC_CONFIG_SRCDIR([src/helloworld.c])
#AC_CONFIG_HEADERS([config.h])

Checks for programs.
AC_PROG_CC

Checks for libraries.

Checks for header files.

Checks for typedefs, structures, and compiler characteristics.

Checks for library functions.

AC_OUTPUT
```

# Autoconf & Automake

- Create the macros for automake.
- These will be in aclocal.m4
- Command to execute = aclocal
- Note: make sure you have all permissions on autom4te.cache...

```
peter@PC1:~/CS2/H7-test2$ aclocal
peter@PC1:~/CS2/H7-test2$ ls
aclocal.m4 configure doc man
autom4te.cache configure.ac examples scripts
autoscan.log configure.ac.first Makefile.am src
peter@PC1:~/CS2/H7-test2$
```



# Autoconf & Automake

- Produce Makefile.in from Makefile.am and see what's missing...
- Automake --add-missing

```
peter@PC1:~/CS2/H7-test2$ automake --add-missing
configure.ac:6: warning: AM_INIT_AUTOMAKE: two- and three-argument
s forms are deprecated. For more info, see:
configure.ac:6: http://www.gnu.org/software/automake/manual/automake.html#Modernize-AM_005fINIT_005fAUTOMAKE-invocation
configure.ac:12: installing './compile'
configure.ac:6: installing './install-sh'
configure.ac:6: installing './missing'
src/Makefile.am: installing './depcomp'
peter@PC1:~/CS2/H7-test2$
```

# Autoconf & Automake

---

- Run autoconf to create the full configuration script.

```
peter@PC1:~/CS2/H7-test2$ autoconf
peter@PC1:~/CS2/H7-test2$ ls
aclocal.m4 configure doc Makefile.in src
autom4te.cache configure.ac examples man
autoscan.log configure.ac.first install-sh missing
compile depcomp Makefile.am scripts
peter@PC1:~/CS2/H7-test2$
```

# Autoconf & Automake

- Now you can run:
- ./configure
- make
- sudo make install
- ... and the binary will be created in /src...

```
peter@PC1:~/CS2/H7-test2$ cd src
peter@PC1:~/CS2/H7-test2/src$ ls
helloworld helloworld.o Makefile.am
helloworld.c Makefile Makefile.in
peter@PC1:~/CS2/H7-test2/src$ helloworld
Hello world!
peter@PC1:~/CS2/H7-test2/src$
```

---

# Computersystemen 2

## Labo

## Compileren

---

# Inhoud

- Compileren
- Make

---

# Inhoud

- Compileren

# Compileren

---

- `gcc -o programma programma.c`
- `g++ -o programma programma.cpp`
- ... zie theorie gedeelte.

---

# Inhoud

- Make



# Source files

---

- 4 files:
  - Main.cpp = source file
  - Helloworld.cpp
  - Factorial.cpp
  - Functions.h = header file

# Source files

- 4 files:

```
#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;
}
```

- **Main.cpp** = source file
- Helloworld.cpp
- Factorial.cpp
- Functions.h = header file

# Source files

---

- 4 files:
  - Main.cpp = source file
  - Helloworld.cpp
  - Factorial.cpp
  - Functions.h = header file

```
#include <iostream>
#include "functions.h"
void print_hello(){
std::cout << "Hello World!";
}
```

# Source files

---

- 4 files:
  - Main.cpp = source file
  - Helloworld.cpp
  - Factorial.cpp
  - Functions.h = header file

```
#include "functions.h"
int factorial(int n){
 if(n!=1){
 return(n * factorial(n-1));
 }
 else return 1;
}
```

# Source files

---

- 4 files:
  - Main.cpp = source file
  - Helloworld.cpp
  - Factorial.cpp
  - Functions.h** = header file

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

# Manueel compileren...

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

## Compile met make...

- Sudo apt-get install make
- make zoekt naar  
"Makefile" in source  
bestand

# Compile met make...

- Makefile: bevat compileer-instructies.
- Basis:  
target: dependencies  
[tab] systeemcommando's

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



# Compile met make...

- Makefile Met dependencies...

```
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
```

# Compile met make...

- Makefile Met variabelen en

```
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
```

Wat is er anders tov vorige

# Compile met make...

- Makefile Met variabelen en

```
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

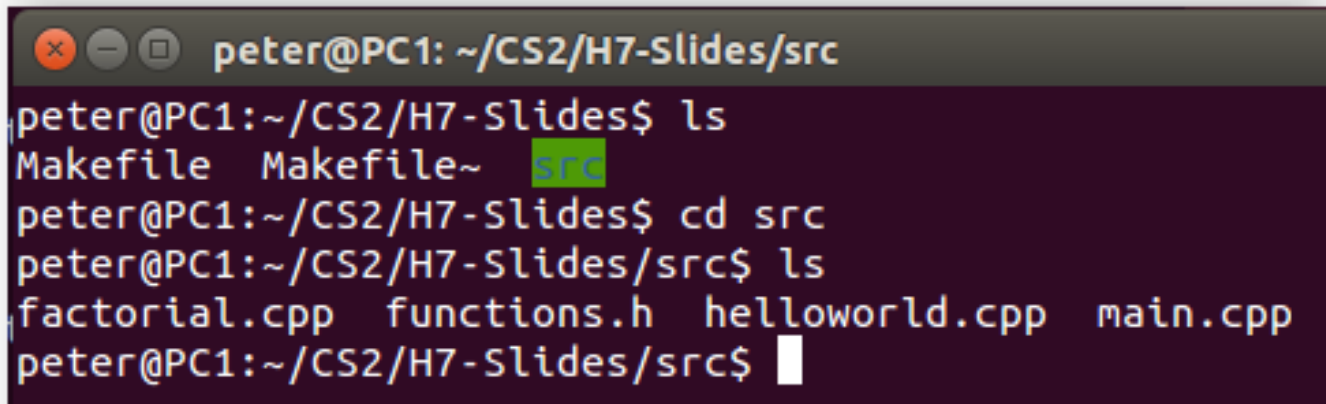
helloworld.o: helloworld.cpp
 $(CC) $(CFLAGS) helloworld.cpp

clean:
 rm -rf *.o helloworld
```

Wat is er fout tov vorige slide?  
De /src zijn verdwenen...  
(Gaat er dus van uit dat de sources in de huidige dir

# Compile met make...

- Makefile Met variabelen en commentaar
- We testen of het werkt:



```
peter@PC1: ~/CS2/H7-Slides/src
peter@PC1:~/CS2/H7-Slides$ ls
Makefile Makefile~ src
peter@PC1:~/CS2/H7-Slides$ cd src
peter@PC1:~/CS2/H7-Slides/src$ ls
factorial.cpp functions.h helloworld.cpp main.cpp
peter@PC1:~/CS2/H7-Slides/src$
```

# Compile met make...

- Makefile Met variabelen en commentaar
- We testen of het werkt:

```
g++ -c -Wall src/main.cpp
g++ -c -Wall src/factorial.cpp
g++ -c -Wall src/helloworld.cpp
g++ main.o factorial.o helloworld.o -o helloworld
peter@PC1:~/CS2/H7-Slides$
```

# Compile met make...

- Makefile Met variabelen en commentaar
- We testen of het werkt:
  - Commando = make
  - Optioneel Commando = make helloworld

```
peter@PC1:~/CS2/H7-Slides$ ls
factorial.o helloworld helloworld.o main.o Makefile Makefile~ src
peter@PC1:~/CS2/H7-Slides$
```

# Compile met make...

- Makefile Met variabelen en commentaar
- We testen of het werkt:

```
peter@PC1:~/CS2/H7-Slides$ ls
factorial.o helloworld helloworld.o main.o Makefile Makefile~ src
```

```
peter@PC1:~/CS2/H7-Slides$./helloworld
Hello World!
De faculteit van 10 is 3628800
peter@PC1:~/CS2/H7-Slides$
```

# Manpage

- Debian Policy: manpage nodig voor elke executable.
- Genereren met gmanedit
- Install: `sudo apt-get install gmanedit`
- Gebruik: **sudo** gmanedit
  - Anders gewone texteditor

Zie:

<http://www.ghacks.net/2010/06/04/edit-linux-man-pages-with-gmanedit/>



# Manpage

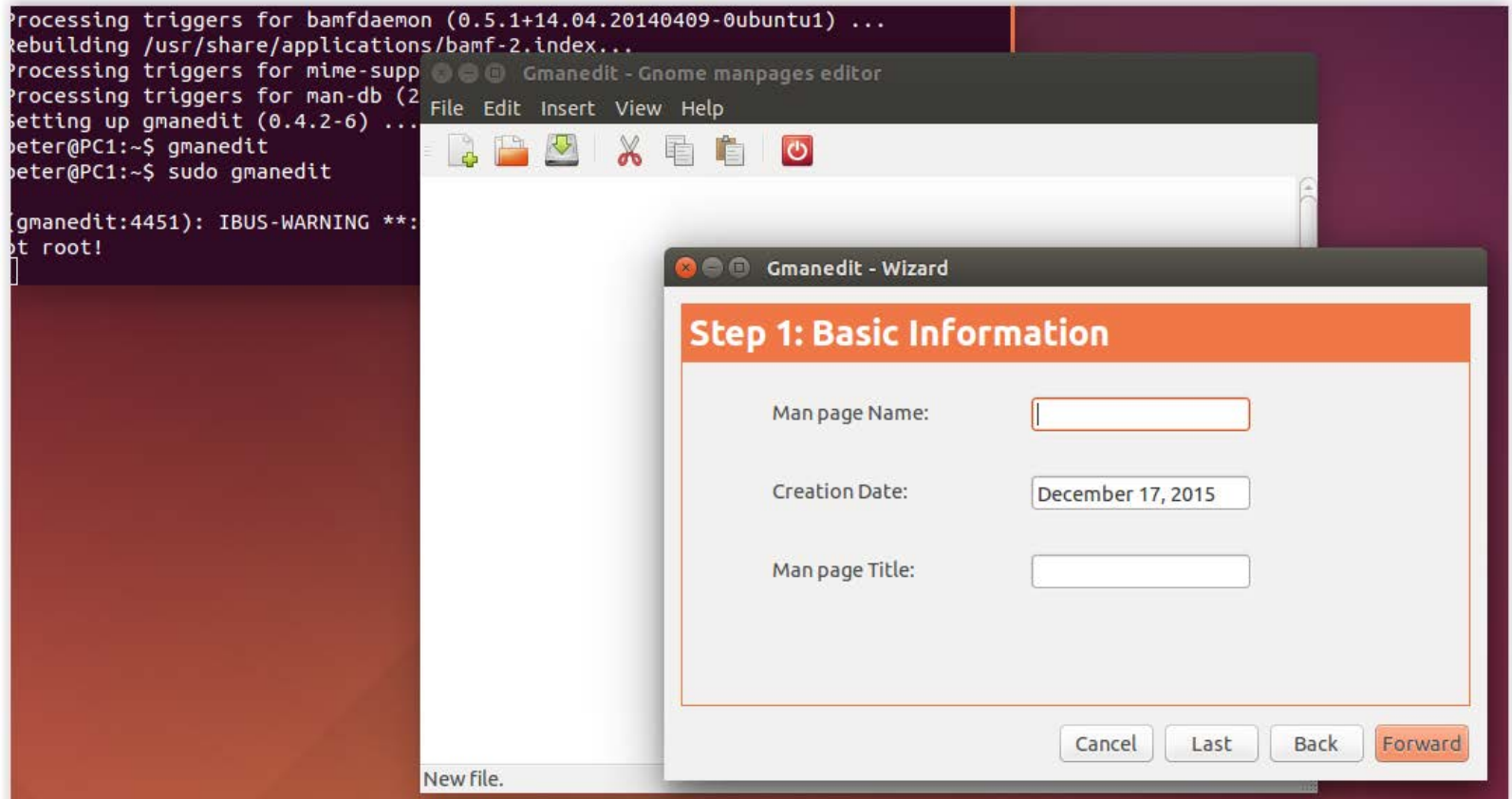
---

- Manpage sections:

| 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 |

# Manpage

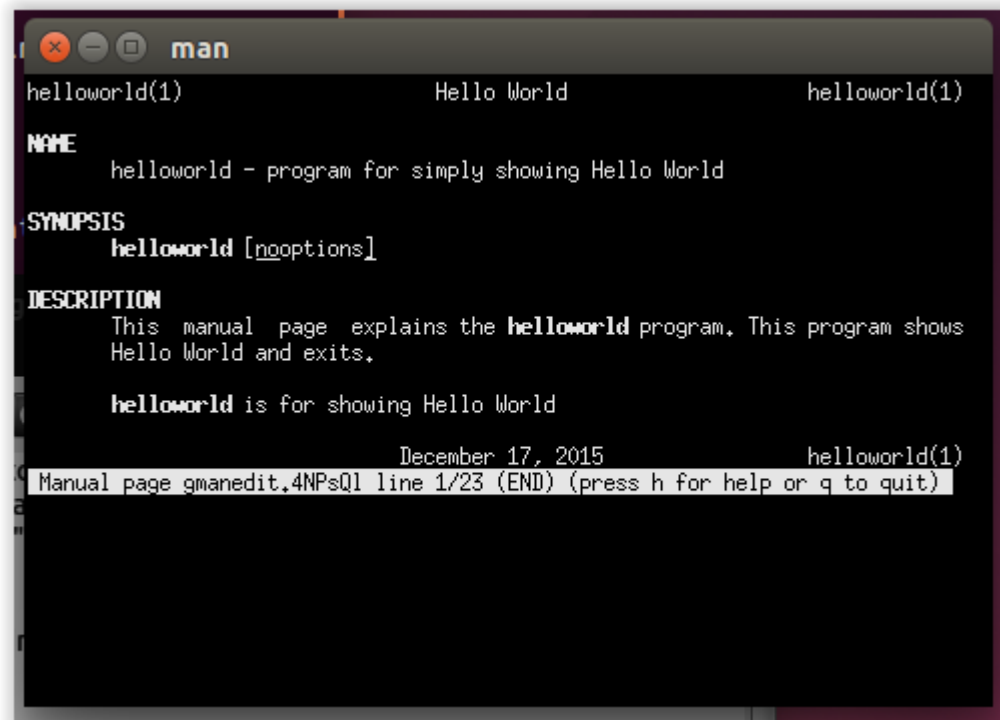
- Manpage maken:
  - Wizard



# Manpage

---

- Manpage maken: "View Created Page"



```
man
helloworld(1) Hello World helloworld(1)

NAME
 helloworld - program for simply showing Hello World

SYNOPSIS
 helloworld [noptions]

DESCRIPTION
 This manual page explains the helloworld program. This program shows
 Hello World and exits.

 helloworld is for showing Hello World

 December 17, 2015 helloworld(1)
Manual page gmanedit.4NPsQ1 line 1/23 (END) (press h for help or q to quit)
```

# Manpage

- Manpage maken: (vb)

```
.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
```

# Manpage

- Locatie:
  - The system stores its man pages at `/usr/share/man/` directory as described in about section. For example, the directory `/usr/share/man/man1` stores man pages for user shell commands.
  - It is recommended that you store your own man pages in `/usr/local/man` directory.

```
/usr/local/share/man/man1/helloworld.1
peter@PC1:~/CS2$
```

# The end...

---