Linux Labo 15
Herhaling

Basic Terminal commando’s

man

Man is het ingebouwde documentatie systeem van linux.

Als je meer informatie wil hebben over een bepaald commando dan kan dat steeds m.b.v. man

$> man [commando]

Bvb $> man man → geeft meer info over man zelf

Directory Structuur

Wat is een directory?

Bestanden in de Terminal vind je terug in directories.

Directories zijn de mapjes van je grafische omgeving

Deze directories zijn geordend volgens een boomstructuur. Toepasselijk wordt het éérste niveau de “root” of wortel genoemt.

Directory Structuur

Hoe ziet het eruit?

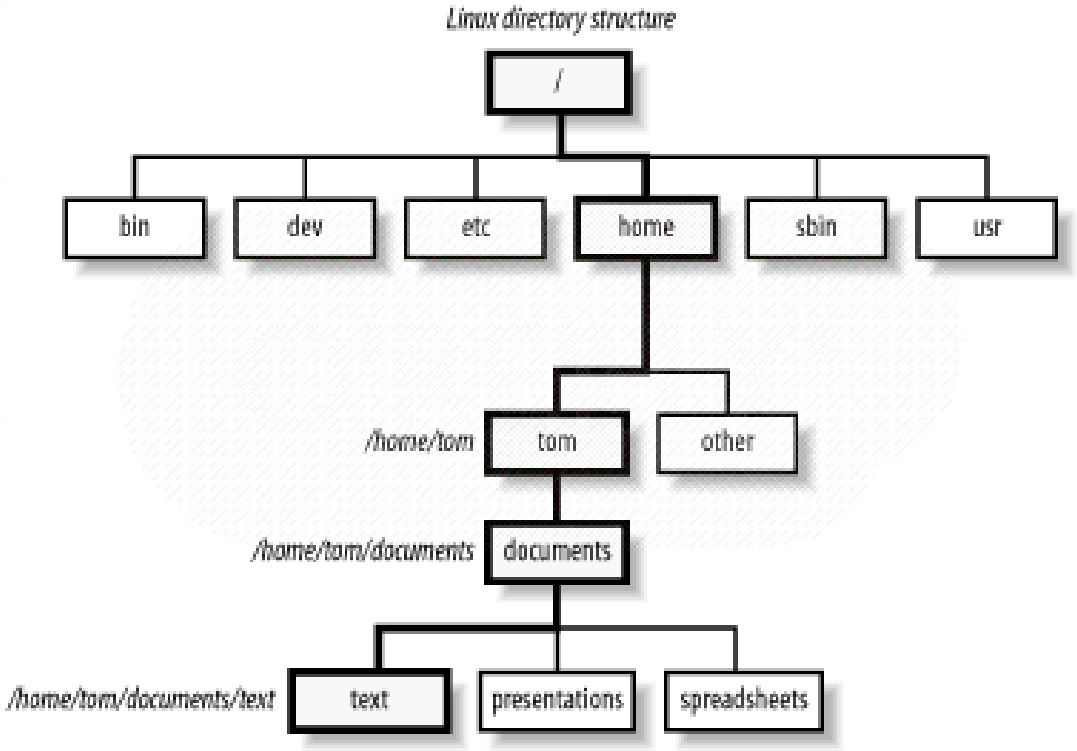
De root wordt aangeduid met enkel een “/” en alle respectivelijke niveau’s daaronder worden ook met behulp van een “/” als separator aangegeven.

/
/home /bin /lib /etc ...
/student

/home/student $>

Directory Structuur

Hoe ziet het eruit?



Basic Terminal commando’s

ls

Om een lijst te krijgen van de bestanden in de actuele directory geeft men het commando ls

$> ls [opties][path]

$> man ls → geeft meer info over het ls commando

$> ls -1 → geeft de lijst files onder elkaar
$> ls -a → toont alle files ook de verborgen bestanden
$> ls -l → geeft een gedetaileerde lijst, files onder elkaar

Basic Terminal commando’s

ls

Als je geen “path” meegeeft gaat het om de “working directory” of de plaats in de boomstructuur waar je bent.

$> ls [opties][path]

$> ls /bin
Geeft de bestanden in de directory bin

Basic Terminal commando’s

cd

Met cd kan men op en neer bewegen in de boom structuur van het file systeem.

$> cd [directory]

$> cd / → verplaatst de prompt naar de root van het file systeem

$> cd var → verplaatst de prompt naar de var directory

$> cd .. → verplaatst de prompt één directory dichter bij root.

$> cd → verplaatst de prompt naar de home directory.

Basic Terminal commando’s

pwd

Dit staat voor “print working directory” en toont je waar je precies bent in de boomstructuur.

$> pwd

$> pwd

/home/student/mijndirectory

Basic Terminal commando’s

cp

Met cp kan je een bestand kopiëren.

$> cp [bronbestand] [naamkopie]

$> cp file1 file2

file2 is nu een kopie van file1

Basic Terminal commando’s

mv

Met mv kan je een bestand verplaatsen.
Een bestand van naam veranderen is ook een soort ‘verplaatsen’

$> mv [oorsprong] [bestemming]

$> mv file1 directory1
file1 staat hierna in directory1

$> mv file1 file2
file1 heeft hierna een nieuwe naam nl. file2

Basic Terminal commando’s

rm

Met rm kan je een bestand wissen.
Pas op met de opties van dit commando !

$> rm [opties] [bestand]

$> rm file1
wist de het bestand file1

$> rm -r directory1
wist directory1 en ook ALLE bestanden erin !

Basic Terminal commando’s

mkdir

Met mkdir creëer je een nieuwe directory. Dat doe je in de working directory

$> mkdir [path][directorynaam]

$> mkdir mijndir
Nieuwe directory in de working directory
$> mkdir /home/student/mijndir
Nieuwe directory in het aangegeven path

Basic Terminal commando’s

rmdir

Met rmdir verwijder je een nieuwe directory. Dat gaat enkel als deze geen bestanden bevat.

$> rmdir [path][directorynaam]

$> rmdir mijndir
‘mijndir’ in de working directory wordt gewist
$> rmdir /home/student/mijndir
Wissen ‘mijndir’ in het aangegeven path

Basic Terminal commando’s

touch

Met touch creëer je een nieuw leeg bestand. Als je het touch commando geeft op een bestaande bestandsnaam blijft het bestand behouden maar de datum en tijd wordt aangepast naar nu.

$> touch [opties][bestandsnaam]

$> touch mijnbestand

Extra Terminal commando

tree

→ Dit commando moeten we installeren ←
Tree toont op een iets grafischer manier de inhoud van de directory structuur.

$> tree [opties][path]

$> tree -d /home/student/labo
Toont enkel de directories

Beperk het gebruik van dit commando. Het is een hulpmiddel om de directory-tree beter te begrijpen

Collectie Selectors

*

Wanneer je een * gebruikt als bestands naam dan begrijpt de computer dit als “alle”

*v.b. $> mv * /dir5 → verplaatst alle files naar /dir5*

Collectie Selectors

*

Wanneer je * gebruikt als deel van een naam dan gaat het om alle bestanden die starten of eindigen met het deel die je opgaf

v.b. $> mv tekst /dir5 → verplaatst alle files die starten met het woord tekst naar /dir5*

Collectie Selectors

?

Wanneer je ? gebruikt als deel van een naam wil dit vraagteken zeggen ‘om het even welk teken’ maar slechts één teken

v.b. $> mv ?ekst /dir5 → verplaatst bestanden die starten met om het even welke letter en eindigen op “ekst”

Je kan meerdere “?” gebruiken bvb *“??ks?”*

Path Completion

~

Als je een path start met **~** (tilde) is dit hetzelfde als het path starten met je home directory

v.b. $> cp file1 ~/file2 → maakt 'n kopie van 'file1', plaatst deze in de home directory van de ingelogde gebruiker en noemt deze kopie 'file2'

Path Completion

.

Als je een path completeert met . (punt) is dit als zeggen “dezelfde naam”. Met andere woorden krijgt de ‘bestemming’ dezelfde naam als de bron

v.b. $> cp file1 /dir/. → *maakt ‘n kopie van ‘file1’, plaatst deze in de directory ‘dir’ en geeft deze kopie dezelfde naam dus ook ‘file1’*

Path Completion

.

Deze punt werkt niet enkel met een bestandsnaam
maar eveneens met een volledig path
of gewoon een deel ervan

v.b. $> cp /dir1/a/b/c/d/e/f/file1 /dir2/.

Zal een kopie maken van file1,
plaatsen in dir2/a/b/c/d/e/f (als deze bestaat...)
en deze eveneens file1 noemen

Path Completion

[TAB]

Wanneer je path en/of bestandsnamen moet typen kan je steeds gebruik maken van de magische Path completion knop [TAB]

stel er staat een file ‘ditiseenfilemethelelangenaam’ in de actuele directory en geen enkele andere file start met ‘ditis’ dan is volgende typen genoeg:

v.b. $> ls -l ditis[TAB]

Linux Users & Groups

useradd

Linux is een multi-user omgeving

Met useradd kunnen we een user bij-creëren

$> useradd [options] [user]

Linux Users & Groups

passwd

Met passwd kan je het paswoord van

een bepaalde user instellen of wijzigen

$> passwd [username]

Linux Users & Groups

adduser

Doet hetzelfde als useradd maar maakt eveneens een mapje aan in de /home directory voor deze user, vraagt meteen om een paswoord etc.

$> adduser [options] [user]

Linux Users & Groups

userdel

Spreekt voor zich,
met userdel kunnen we een user terug verwijderen

$> userdel [option] user

Linux Users & Groups

su

Met het su commando kan je tijdelijk een andere identiteit aannemen, m.a.w ‘inloggen’ onder een andere usernaam

$> su [usernaam]

Met **“exit”** kom je terug in de vorige identiteit terecht

Linux Users & Groups

whoami

Als je twijfelt welke user-name je nu precies als actuele identiteit hebt kan je het whoami commando geven

$> whoami [options]

Linux Users & Groups

“su do”

Als gewone gebruiker heb je niet genoeg rechten om gebruikers of groepen aan te maken

Enkele de ‘root’-administrator heeft deze rechten

(Let op ‘root’ is hier niet hetzelfde als ‘/’ !)

Linux Users & Groups

sudo -i

Als je ‘sudo -i’ geeft dan neem je net als bij ‘su’ een andere identiteit aan.

Na dit commando wordt je “root”

Met “**exit**” keer je terug naar je vorige identiteit

Het is echter best om énkel root te zijn wanneer dit écht nodig is, immers als je als root een fout maakt kan dit desastreuze gevolgen hebben...

Linux Users & Groups

sudo

Om te voorkomen dat je telkens moet uitloggen als gewone gebruiker en terug inloggen als ‘root’ werkt het sudo commando ook voor één enkele opdracht

$> sudo [commando]

voert het command uit als root

Voorbeeld: $> sudo adduser

Linux Users & Groups

groupadd

Users kunnen in groepen onderverdeeld worden

Met groupadd kunnen we een groep bij-creëren

$> groupadd [options] group

Linux Users & Groups

groupdel

Spreekt opnieuw voor zich,
met groupdel kunnen we een groep verwijderen

$> groupdel group

Linux Users & Groups

usermod

Met user mod kunnen we aanpassingen doen aan de instelling van een user

bvb een user toevoegen aan een groep

$> usermod -a -G [group] [user]

(oefening: lees de man page voor alle opties)

Linux Users & Groups

groupmod

Met groupmod kunnen we aanpassingen doen aan bestaande groepen

Bvb de naam van de groep aanpassen:

$> groupmod -n [nieuwenaam] [oudenaam]

Linux Users & Groups

groups

Met groups kunnen we zien
van welke groepen een user allemaal lid is

$> groups [user]

Linux Users & Groups

members

→ Dit commando moeten we installeren ←
Met het commando ‘members’ kijken welke users er allemaal lid zijn van een bepaalde groep

$> members [group]

Linux Users & Groups

id

Met het commando id kunnen we
wat diepere informatie uit het systeem halen
over een bepaalde gebruiker zoals o.a. ‘uid’ en ‘giu’

$> id [options][user]

Linux gebruikt intern namelijk *nummers* i.p.v
namen voor gebruikers en groepen.

Linux File rechten systeem

chmod

In linux kan je bepalen wie er wel of niet een bestand mag lezen, bewerken of uitvoeren.

Dit kunnen we aanpassen met ***chmod.***

a) met nummers: n = 0..7

$> chown [nnn] [bestandsnaam]

Bvb: chown 640 mijnfile.txt

Linux File rechten systeem

chmod

Basis:

Uitvoeren → execute = “x” = 1
Schrijven → write = “w” = 2
Lezen → read = “r” = 4

Het resultaat is een getal tussen 0 en 7

bvb lezen + uitvoeren = 1 + 4 = 5

Linux File rechten systeem

chmod

Verder zijn er permissie categorieën:

Een bestand heeft een eigenaar,
=> er zijn de rechten van de ***eigenaar***

Een bestand is onderdeel van een groep
=> er zijn de rechten van de ***groep***

Een bestand staat in het file systeem
=> er zijn de rechten van ***alle*** users

Linux File rechten systeem

File attributes

drwxrwxrwx

Elke bestand heeft bestands attributen

d → staat voor file of directory

Eerste “rwx” → staat voor rechten v/d eigenaar
Tweede “rwx” → staat voor rechten v/d groep
Derde “rwx” → staat voor rechten van alle users

Linux File rechten systeem

chmod +/-

b) chmod met letters:

+r → lezen toestaan
-r → lezen blokkeren

+w → schrijven toestaan
-w → schrijven blokkeren

+x → uitvoeren toestaan
-x → uitvoeren blokkeren

Linux File rechten systeem

chmod +/-

Om aan te geven of het gaat over de eigenaar, de groep, alle anderen, of over iedereen (all) gebruiken de vorige zaken samen met de volgende letters:

u → user (eigenaar)
g → group (groep)
o → other (alle anderen)
a → all = user + group + other

Linux File rechten systeem

chmod +/-

Met enkele voorbeelden wordt dit meteen duidelijker:

$> chmod u+x bestand1

→ user kan nu uitvoeren

$> chmod g-w bestand2

→ groep kan niet meer schrijven

Linux File rechten systeem

chown

Met het **chown** commando kan je aanpassen wie de eigenaar is van een bestand.

$> chown [usernaam]:[groep] [bestandsnaam]

(Let op de dubbele-punt tussen user en group !)

Linux Users & Groups

deluser

We zagen hoe we een user kunnen verwijderen met ‘userdel’, er is echter één speciale versie:

deluser [user] [group]

Hiermee kunnen een user verwijderen
uit één bepaalde groep.

De user blijft hier bestaan
(op voorwaarde dat je user als groep opgeeft)

Linux File rechten systeem

cd

We zagen reeds wat ‘cd’ *doet*,

belangrijk is te ook weten wat ‘cd’ *is* :

‘cd’ is eigenlijk een *programma* dat we *uitvoeren*.

Linux File rechten systeem

cd

d.w.z dat als het we het *programma*

cd

willen aanroepen op een directory

deze directory ‘*uitvoer-rechten*’ moet hebben

Linux File rechten systeem

cd

Enkele voorbeelden:

drwxrwxrwx → zowel de eigenaar,
als de groep,
als anderen kunnen
‘cd’ uitvoeren

drwxrw-rw- → énkel de eigenaar,
kan ‘cd’ uitvoeren

Linux File rechten systeem

ls

We zagen eveneens reeds wat ‘ls’ doet

→ *Merk op*: Je kan ook een of meerdere bestandsnamen opgeven bij ‘ls’

Voorbeeld:

$> ls -al file1.txt

Linux Backup Commands

gzip

Met zip kunnen we een bestand comprimeren
Volgens het gzip formaat

$> gzip [file]

Bvb: $> gzip -k tekst.txt
-k → zip met behoud van bronbestand

Linux Backup Commands

gunzip

Met gunzip kunnen we een bestand terug de-comprimeren

$> gunzip [file]

Bvb: $> gunzip tekst.txt.gz

Linux Backup Commands

tar

Met ‘tar’ kunnen we een archief bestand maken van een reeks andere bestanden. Afhankelijk van de opties kunnen we ook compressie toevoegen waardoor het archief kleiner wordt dan de originele files

$> tar [options] [targetfile] [sourcefile]...

Linux Backup Commands

tar

tar is een “tweerichtings” commando, van files naar een archief en van een archief terug naar die files

De opties zijn als volgt:

-c = “create” → archief maken
-x = “extract” → de-archiveren
-v = “verbose” → toont wat tar aan het doen is

Linux Backup Commands

tar

-r = “append” → voegt files toe aan een bestaand archief

-f = “file” → gebruik een archief bestand (altijd)

-z = “g(un)zip” → comprimeer het archief met gzip

Linux Backup Commands

rsync

Met rsync kunnen we een bestand of directory synchroniseren met een ander bestand of directory

Bvb enkel de bestanden die veranderd zijn worden vervangen (of gewist → optie)

$> rsync [options] [source][target]

Linux Backup Commands

rsync

gebruik:

$> rsync bronbestand doelbestand

$> rsync -vrut brondirectory/ doeldirectory/

Linux Stream redirection

>

In linux kunnen we de tekst output van een programma nemen en naar ergens anders sturen dan het beeldscherm

$> [command] > [target]

Bvb $> ls > lijst.txt

Stuurt de output van ls naar bestand lijst.txt

Linux Stream redirection

>>

Hetzelfde als de vorige redirection maar via ‘append’
in plaats van bestand creatie.

$> [command] >> [target]

Bvb $> ls >> langelijst.txt

plakt de output van ls onderaan het bestand lijst.txt

Linux Text Commands

echo

Met echo kunnen we iets weergeven op het scherm

Door “redirection” kunnen we zo bvb tekst naar een bestand schrijven

$> echo [options] [textstring]

Linux Text Commands

cat

Met cat kunnen we de inhoud van een bestand weergeven in een terminal

$> cat [options] [filename]

Linux Text Commands

head

Head doet hetzelfde als cat maar geeft slechts de éérste aantal lijnen weer.

Default is dat 10 maar via opties kunnen we dit aanpassen naar meer of minder

$> head [options] [filename]

bvb $> head -20 tekst.txt

Linux Text Commands

tail

Tail doet hetzelfde als head maar geeft in plaats van de éérste aantal lijnen nu enkel de laatste lijnen weer.

Default is ook dat 10 maar via opties kunnen we dit opnieuw aanpassen naar meer of minder

$> tail [options] [filename]

bvb $> tail -6 tekst.txt

Linux Text Commands

more

Met ‘more’ kunnen we de tekst in pagina’s onderbreken tijdens het weergeven. Met de spatiebalk gaan we naar de volgende pagina.

$> more [options] [bestand]

Met de letter “q” kunnen we more verlaten

Linux Text Commands

less

Met ‘less’ kunnen we de tekst weergeven op een manier die scrollen ondersteunt. Met de pijltjes up en down lopen we door de tekst.

$> less [options] [bestand]

Met de letter “q” kunnen we less verlaten

Linux Text Commands

grep

Zoekt het opgegeven patroon in het opgegeven bestand en drukt de lijnen af waar dit patroon voorkomt.

$> grep [options] [patroon] [bestand]

Bv $>grep hallo halloworld.txt

hallo world

Linux Text Commands

diff

Zoekt het verschil tussen twee bestanden
en drukt enkel de lijnen af die verschillend zijn

$> diff [options] [bestand1] [bestand2]

Bv $>diff tekst1.txt tekst2.txt

5d4 →*indicatie waar verschil zit*
< verschil →*tekst die verschillend is*

Linux Text Commands

clear

Met “control-L” kunnen we normaal alle tekst in een terminal wegdoen zodat we een propere lei hebben.

‘clear’ is een commando dat net hetzelfde doet

Linux Text Commands

nano

De meeste instellingen in linux gebeuren door middel van gewone tekstbestanden.

Met echo en stream redirection
kan je wel een tekstbestand creëren maar achteraf iets aanpassen is geen evidentie …

Er is dus nood aan ‘tekst editors’
Dit zijn ‘mini tekst-verwerkers’

Linux Text Commands

nano

Nano gebruiken:

$> nano [opties][bestandsnaam]

Belangrijkste commando's

Ctrl + x → afsluiten
Ctrl + o → bewaren document
Ctrl + w → zoeken in tekst
Ctrl + k → lijn wissen
Ctrl + u → lijn terug toevoegen

Command stream redirection

|

Met het ‘pipe’ redirection symbool kunnen we de output van één commando direct naar het volgende commando sturen als input

$> [command] | [command]

Bvb $> ls | more

De output van ‘ls’ wordt de input van ‘more’

Opeenvolgende commando’s : concatenatie

;

Door meerdere commando’s op één lijn te schrijven
en deze met een punt-comma te scheiden kunnen
we die commando’s opeenvolgend laten uitvoeren
door de interpreter

$> [command] ; [command] ; ...

Bvb $> clear; echo bestanden-lijst; ls

Opeenvolgende commando’s : concatenatie

&&

Dit is ook een concatenatie maar conditioneel. Het volgende commando wordt énkel uitgevoerd als het eerste commando zonder fouten beëindigde.

$> [command] && [command]

Bvb $> mkdir test && echo gelukt!

Opeenvolgende commando’s : concatenatie

||

Dit is ook een conditionele concatenatie. Het volgende commando wordt énkel uitgevoerd als het eerste commando *mét* fouten beëindigde.

$> [command] || [command]

Bvb $> mkdir test || echo mislukt...

Opeenvolgende commando’s : scripts

#!/bin/bash

Om een bash script te hebben moet een tekstbestand voldoen aan 2 voorwaarden:

1) het moet starten met de zin: #!/bin/bash

2) het bestand moet uitvoerrechten krijgen.

Opeenvolgende commando's : scripts

./

Om de interpreter een script te laten uitvoeren moet men ofwel de naam van het bestand opgeven inclusief z'n volledige path

Alternatief kan men het script starten door de naam van het bestand op te geven voorafgegeaan door "punt-slash"

Bvb $> ./mijnscript.sh

Opeenvolgende commando’s : scripts

bestandsnaam.sh

Het is gebruikelijk om bash scripts de extentie “.sh” mee te geven.

Dit is niet noodzakelijk, linux gebruikt immers geen extenties, maar maakt het voor de gebruiker gemakkelijker om scripts te herkennen.

Variabelen in scripts & bash

Principe (cijfers):

$> mijnvar=1234

$> echo $mijnvar

Variabelen in scripts & bash

Principe (tekst):

$> mijnvar=”tekst”

$> echo $mijnvar

Variabelen in scripts & bash

Speciale variabelen:

“environment variables”

PS1, PS2,
PATH, HOME,
REPLY (zie “read”)
...

Variabelen in scripts & bash

printenv

Printenv drukt via ‘echo’ alle
“environment variables”
af in de terminal

$>printenv

data input in scripts

read

Met read kunnen we de user om gegevens vragen.
Het resultaat slaan we op in een variabele.

read [options][variable1 variable2 …]

$> read -p “wat is je naam? ” naamvar
$> read → resultaat in “$REPLY” var

data input in scripts

read

Opties:

-p → <prompt> : echo de tekst na de -p
-d → <delimitor> : kiest een ander teken ipv spatie als tussen variabelen
-n → <x> : leest x chars als input
-s → ‘secure’ : verbergt de ingave

Conditionals in scripts

if then fi

We kunnen er voor zorgen dat bepaalde commando's slechts uitgevoerd worden als er aan een voorwaarde voldaan is

```
if [  "a" = "a"  ]; then
        echo "de letters zijn gelijk"
fi
```

Conditionals in scripts

if then else if

Het else commando geeft de mogelijkheid dingen uit te voeren énkel als niét aan de conditie voldaan is

```
if [  “a” = “a”  ]; then
        echo “de letters zijn gelijk”
else
        echo “de letters zijn niet gelijk”
fi
```

Conditionals in scripts

Bij gebruik cijfers:

-eq → Equal to
-ne → Not equal to
-lt → Less than
-le → Less than or equal to
-gt → Greater than
-ge → Greater than or equal to

If [$a -lt 10] → als $a kleiner is dan 10 doe dan

Conditionals in scripts

Bij gebruik files:

-e → File exists
-f → File is a *regular* file
-d → File is a *directory*

- r → File has read permission
-w → File has write permission
-x → File has execute permission

If [-e “$file”] → als de file bestaat doe dan

Loops in scripts

for in do done

Soms willen we alle items in een lijst doorlopen, bijvoorbeeld om met elk bestand een bewerking te doen of om er iets in te zoeken.

Hiervoor hebben we in Bash het ‘for’ commando.

Het ‘for’ commando in Bash werkt ietwat anders dan wat jullie reeds kennen van C of C#

Conditionals in scripts

Case in esac

Bij “if then fi” is kunnen we slechts op één conditie testen, hooguit nog uitbreidbaar met “else”.

Vaak kan een ‘vraag’ meerdere ‘antwoorden’ hebben en is een if statement voor elke keuze omslachtig.

Daarom is er het “case” statement !

Het case statement werkt als volgt:

Conditionals in scripts

Case in esac

```
case [expresie] in
      [patroon-1] )
                  … code ...
      ;;
      [patroon-2] )
                  … code ...
      ;;
      [patroon-n] )
                  … code ...
      ;;
esac
```

Commando uitbreidingen

alias

Met het commando alias kunnen we zelf onze eigen commando's creëren in bash.

$> alias [newcommand]=”[code]”

Voorbeeld: $>alias cls=”clear; ls”

$>cls → wist het scherm + ls

Commando uitbreidingen

$0 en $1 $2 $3… $n

Dankzij de variabelen $0 en $1, $2 etc kunnen we bij het starten van ons script direct vanin bash extra parameters meegeven aan een te starten script. $0 is hier de naam van het script zelf.
$1...$n zijn alle volgende parameters

$> ./[scriptnaam] [parameter1] [parameter2] ...

Voorbeeld: $>./mijnscript halloworld

→ *in het script bevat de variabele* ***$1*** *nu de tekst “halloworld”*

Linux Text Commands

tput

Met het commando tput kunnen we enkele terminal instellingen aanpassen, zoals tekstkleur, achtergrond, cursorplaats, …

$> tput [option] [parameter]

Voorbeeld: *$> tput setaf 2*

→ de terminal tekst is nu groen

Linux Text Commands

tput

veelgebruikte opties:

tput setaf [0..9] → *zet de terminal tekst kleur*
tput setab [0..9] → *zet de achtergrond kleur*
tput bold → *maak de terminal tekst bold*
tput dim → *maak de terminal tekst minder helder*
tput sgr0 → *zet alle instelling terug default*
tput cup [x] [y] → *zet cursor op plaats x,y*

Linux Text Commands

tput

Bij de opties “setaf” en “setab” representeren de cijfers van 0 tot 9 volgende kleuren

0	**Black**	**5**	**Magenta**
1	**Red**	**6**	**Cyan**
2	**Green**	**7**	**White**
3	**Yellow**	**8**	**Not used**
4	**Blue**	**9**	**Reset to default color**

Linux Text Commands

#

Met een “#” als éérste character van een lijn kan je commentaar toevoegen aan je script.

```
# deze lijn wordt niet uitgevoerd
# ook deze lijn is commentaar
if [ “a” = “a”]
then
        ….
        ….
```

Loops in scripts

while do done

Vaak willen we delen van ons programma herhalen tot een bepaalde voorwaarde voldaan is.

Hiervoor hebben we ‘loops’.

while [conditie]
do
    … code …
done

Loops in scripts

until do done

Soms willen we code uitvoeren totdat een conditie voldaan is in plaats van zolang dat een conditie voldaan is. De oplossing is het ‘until’ commando.

‘until’ *stopt als de conditie voldaan is* daar waar ‘while’ de code uitvoert *als de conditie voldaan is*

```
until [conditie]
do
    … code …
done
```

Loops in scripts

until do done

Voorbeeld:

```
getal=0

until [ $getal -eq 9 ]
do
        echo "geef je keuze”
        echo “kies '9' om te stoppen"
        read getal
        echo "jou keuze was $getal"
done
```

Loops in scripts

for in do done

Een “for” lus loopt doorheen alle items van een lijst en kent deze een voor een toe aan een variabele.

```
for [variabele] in [lijst]
do
    … code …
done
```

Loops in scripts

lijsten
commando substitutie

Om een lijst te maken kunnen we ook gebruik maken van wat noemt commando substitutie:

Bvb: mijnvar = $(ls)

Het resultaat is dat alle tekst dat het commando ‘ls’ produceert nu toegekend wordt aan de variabele ‘mijnvar’

Loops in scripts

for in do done
commando substitutie

Voorbeeld:

```
for bestand in $(ls)
do
    echo $bestand
done
```

Schermafdruk: elke bestandsnaam die “ls” oplevert wordt onderelkaar afgedrukt op het scherm

Loops in scripts

continue

Bij het commando continue in een lus wordt alle volgende code na het continue commando overgeslagen en gaat de lus meteen verder naar de volgende stap

Loops in scripts

continue

Voorbeeld:

```
for n in {1..5}
do
        if [ $n = 3 ]; then
                continue
        fi
        echo $n
done
```

```
1
2
4
5
```

Loops in scripts

break

Als in een lus het break commando wordt ontmoet dan wordt de lus onderbroken en gaat de code verder meteen na de lus.

Loops in scripts

break

Voorbeeld:

```
for n in {1..5}
do
        if [ $n = 3 ]; then
                break
        fi
        echo $n
done
```

1
2

Functies in scripts

function

Als we grotere scripts moeten maken of bepaalde functionaliteit komt herhaaldelijk terug dan kunnen we ook in Bash functies maken.

```
function [ naam ] {
    [code]
}
```

Functies in scripts

function

Voorbeeld:

```
function opnieuw {
        echo “Altijd maar hetzelfde”
}

for n in {1..10}
do
        opnieuw
done
```

Altijd maar hetzelfde
Altijd maar hetzelfde
Altijd maar hetzelfde
Altijd maar hetzelfde ...

error output redirection

2>

We zagen reeds dat we met “>” en “>>” output redirection konden doen.
De output die normaal op het scherm terecht zou komen wordt dan naar een file geschreven.

Bij redirection met “2>” is het de error-tekst die we naar een bestand sturen