

Linux introductie voor systeembeheerders -
voor MBO niveau 4 en het LPI Linux
Essentials examen: De CLI

D. Leeuw

28 februari 2021

v.0.2.0



© 2020 Dennis Leeuw

Dit werk is uitgegeven onder de Creative Commons BY-NC-SA Licentie en laat anderen toe het werk te kopiëren, distribueren, vertonen, op te voeren, en om afgeleid materiaal te maken, zolang de auteurs en uitgever worden vermeld als maker van het werk, het werk niet commercieel gebruikt wordt en afgeleide werken onder identieke voorwaarden worden verspreid.

Over dit Document

Dit document behandelt Linux voor het middelbaar beroepsonderwijs in Nederland, maar kan breder ingezet worden, daar het gericht is op het behalen van het LPI Linux Essentials examen. De doelgroep is niveau 4 van het MBO, met enige kennis van computers.

Versienummering

Het versienummer van elk document bestaat uit drie nummers gescheiden door een punt. Het eerste nummer is het major-versie nummer, het tweede nummer het minor-versienummer en de laatste is de nummering voor bug-fixes.

Om met de laatste te beginnen als er in het document slechts verbeteringen zijn aangebracht die te maken hebben met type-fouten, websites die niet meer beschikbaar zijn, of kleine foutjes in de opdrachten dan zal dit nummer opgehoogd worden. Als docent of student hoeft je boek niet te vervangen. Het is wel handig om de wijzigingen bij te houden.

Als er flink is geschreven aan het document dan zal het minor-nummer opgehoogd worden, dit betekent dat er bijvoorbeeld plaatjes zijn vervangen of geplaatst/weggehaald, maar ook dat paragrafen zijn herschreven, verwijderd of toegevoegd, zonder dat de daadwerkelijk context is veranderd. Een nieuw cohort wordt aangeraden om met deze nieuwe versie te beginnen, bestaande cohorten kunnen doorwerken met het boek dat ze al hebben.

Als het major-nummer wijzigt dan betekent dat dat de inhoud van het boek substantieel is gewijzigd om bijvoorbeeld te voldoen aan een nieuw kwalificatiedossier voor het onderwijs of een nieuwe versie van Linux Essentials van de LPI. Een nieuw major-nummer betekent bijna altijd voor het onderwijs dat in het nieuwe schooljaar men met deze nieuwe versie aan de slag zou moeten gaan. Voorgaande versies van het document zullen nog tot het einde een schooljaar onderhouden worden, maar daarna niet meer.

Document ontwikkeling

Het doel is door middel van open documentatie een document aan te bieden aan zowel studenten als docenten, zonder dat hier hoge kosten aan verbonden zijn en met de gedachte dat we samen meer weten dan alleen. Door samen te werken kunnen we meer bereiken.

Bijdragen aan dit document worden dan ook met alle liefde ontvangen. Let u er wel op dat materiaal dat u bijdraagt onder de CC BY-NC-SA licentie vrijgegeven mag worden, dus alleen origineel materiaal of materiaal dat al vrijgegeven is onder deze licentie.

De eerste versie is geschreven voor het ROC Horizon College.

Inhoudsopgave

Over dit Document	i
1 Inleiding	1
2 Shell scripting	3
2.1 Waarom scripting?	4
2.2 Hello World - een eerste script	4
2.3 Het starten van scripts	5
2.4 Commentaar	6
2.5 Variabelen	7
2.5.1 Opdracht: Werken met variabelen	10
2.6 if	10
2.6.1 Opdracht: Werken met if	13
2.7 for	14
2.8 while	14
Alfabetische index	15

Hoofdstuk 1

Inleiding

Deze Linux cursus beoogt aan te sluiten bij het Linux Essentials examen van de LPI (Linux Professional Institute). De complete opleiding bestaat uit drie delen in het eerste deel installeren we CentOS als werkstation en leren we het Linux systeem kennen vanuit de grafische interface. In het tweede deel installeren we CentOS en zullen we meer leren over het gebruik van Linux op de command line en in het derde deel installeren we Debian en gaan we Linux inzetten als (web)server en zien we de interactie tussen Linux systemen.

Alle Linux systemen zullen geïnstalleerd worden als virtuele machine op Virtual Box (<https://www.virtualbox.org/>). De keuze voor Virtual Box is genomen omdat het gratis beschikbaar is en gebruikt kan worden op Windows, Mac OS X en Linux. Daarmee maakt niet meer uit welk besturings-systeem u nu gebruikt.

Voor de CentOS machine is 15G vrije schijfruimte nodig en voor het Debian systeem 8G daarmee is er een totaal aan 23 G vrije disk ruimte nodig. Voor elke machine hebben we 2G RAM nodig, dus een totaal van 4G moet vrij beschikbaar zijn.

Hoofdstuk 2

Shell scripting

De meeste besturingssystemen kennen een vorm van scripting. Een scripting taal is een programmeertaal die niet gecompileerd hoeft te worden voordat hij uitgevoerd kan worden. De vertaling naar machinecode vindt plaats op het moment dat het script wordt uitgevoerd. Voor de uitvoer van het script is daarom altijd een interpreter (vertaler) nodig die het script vertaalt naar iets dat de computer snapt.

Er zijn verschillende soorten scripting talen met de daarbij behorende interpreters. Voor Linux systemen zijn de belangrijkste:

- Shell scripting (bash of sh)
- Perl
- PHP
- Python
- Java

Naast een command interpreter is de shell op elk Linux-systemen ook een interpreter van een scripting taal, de zogenaamde shell-scripts. Op Linux systemen is de standaard shell **bash** en op Mac OS X aanwezig, dus als je programma's kan schrijven die de shell begrijpt kan je ze op veel systemen toepassen. In de meest simpele vorm is een shell-script een lijstje met commando's. Door het script op te starten worden de commando's één voor één opgestart, maar er is veel meer mogelijk. In dit hoofdstuk ga je leren hoe je shell-scripts schrijft.

2.1 Waarom scripting?

De kracht van shell scripts is dat je de commando's die je op de command line gebruikt direct kunt toepassen in de software die je schrijft. Misschien kunnen we zelfs stellen dat een shell script een lijstje is van commando's die voor later gebruik in de juiste volgorde in het script zijn geplaatst. De vraag is natuurlijk wel, waarom zouden we dat willen?

De belangrijkste reden is dat als er herhaalde handelingen gedaan moeten worden dat de computer dat nauwkeuriger kan dan een mens. Als je eenmaal een script hebt geschreven dat werkt kan je het meerdere keren achter elkaar laten uitvoeren zonder dat er typefouten ontstaan, terwijl typefouten bij mens de meest voorkomende fout is bij het uitvoeren van een commando.

Als het complexe handelingen zijn is het helemaal handig om dit vast te leggen in een script zodat je zeker weet dat je geen stappen over zal slaan.

En als laatste zijn er soms taken die herhaald uitgevoerd moeten worden op vaste dagen of tijden. Denk hierbij aan bijvoorbeeld backups. Het Linux systeem heeft hiervoor een stukje software dat `cron` heet en dat op gezette tijden een commando kan uitvoeren. Door verschillende commando's in een script te plaatsen en `cron` te vertellen het script uit te voeren kunnen er verschillende handelingen in een keer uitgevoerd worden.

2.2 Hello World - een eerste script

We gaan een bestand maken met de naam `texttthello_world.sh`. Het is een goeie gewoonte om een bestandsnaam te eindigen met een extensie die weergeeft wat voor type bestand het is. Dit is voor Linux niet noodzakelijk, maar voor ons gebruikers is het handig om te weten dat we te maken hebben met een shell-script en dus eindigen we de bestandsnaam met `.sh`.

Om structuur in ons werk te houden maken we een directory scripts aan in onze home directoy.

```
$ cd ~  
$ mkdir scripts  
$ cd scripts
```

Maak met `vi` het bestand `hello_world.sh` aan en plaats in het bestand de volgende tekst:

```
echo "Hello world!"  
exit 0
```

Sluit `vi` af. We hebben nu een bestand met 2 commando's. Regel 1 zegt beeld Hello World! op het scherm af en regel 2 zegt verlaat dit script. Linux

leest echter geen extensies om te bepalen wat het moet doen, maar het leest de eerste twee characters van een bestand om te bepalen wat er met een bestand gedaan moet worden. Deze eerste twee characters van een bestand heten het magic number.

Het `file` commando kan gebruikt worden om het magic nummer van een bestand te lezen en weer te geven wat voor soort bestand het is.

```
$ file hello_world.sh
hello_world.sh: ASCII text
```

We zien dat Linux nog denkt dat het om een text bestand gaat. Het ziet het niet als een shell script. Voeg nu boven de `echo` regel een regel toe zodat je script er zo uit komt te zien:

```
#!/bin/bash
echo "Hello world!"
exit 0
```

Als je de editor verlaten hebt en weer een `file` doet van `hello_world.sh` dan zal je zien dat Linux nu weet dat het om een shell-script gaat.

Voor een script dat geïnterpreteerd moet worden door een interpreter, dat kan een shell zijn, zijn de 2-character codes de she-bang: `#!` of hexadecimaal `0x23 0x21`. Achter de she-bang komt de interpreter die het script moet interpreteren. Als de shell deze character reeks tegen komt zal het de rest van de regel lezen om te bepalen aan welke interpreter (shell) het het script moet voeren. In ons voorbeeld wordt het script gegeven aan het programma `/bin/bash` en deze zal de twee opgegeven commando's uitvoeren.

Eerst wordt door `echo Hello world!` op het scherm gezet en daarna sluit het script af. Het `exit` commando is niet noodzakelijk, als de shell geen commando's meer tegen komt zal het automatisch afsluiten. Het voordeel van `exit` is dat je er een exit code aan mee kan geven zodat je kan laten weten dat het commando goed (0) of fout (1) geëindigd is. Doe je dit niet dan is de exit code de error-code van het laatst uitgevoerde commando.

2.3 Het starten van scripts

Het script kan op twee verschillende manieren opgestart worden. Allereerst kunnen we script rechtstreeks aan de shell geven:

```
$ bash ./hello_world.sh
```

De tweede mogelijkheid is dat we het script executable maken en daarna direct uitvoeren. We kunnen een script executable maken voor de eigenaar door het x-bit te zetten voor de user:

```
$ chmod u+x ./hello_world.sh
```

Daarna kan de eigenaar het script direct uitvoeren

```
$ ./hello_world.sh
```

Bij beide opties is het van belang om het complete pad mee te geven aan het script. In de voorbeelden konden we volstaan met `./` omdat het script in de directory staat waar we al in staan. Maar we zouden ook het volledige pad kunnen opgeven:

```
$ /home/dennis/scripts/hello_world.sh
```

We kunnen ook het pad `/home/dennis/scripts` toevoegen aan onze `PATH` variabele en er zo voor zorgen dat we nooit meer het pad hoeven op te geven.

```
$ export PATH=$PATH:/home/dennis/scripts
$ hello_world.sh
```

2.4 Commentaar

Op het moment dat je een programma schrijft is het meestal volledig duidelijk wat je aan het doen bent en waarom. Als het script complexer wordt is het soms al lastiger en als je er na een jaar naar kijkt heb je soms geen idee meer waarom je het hebt gemaakt en hoe het ook alweer in elkaar zat. Het is daarom goed om uitleg in je script te verwerken. Dit maakt het voor jezelf duidelijker, maar ook als iemand anders iets wil wijzigen aan wat jij gemaakt hebt maakt het het voor die persoon een stuk makkelijker als hij of zij weet wat er waar gebeurt in het script. Open met `vi` het `hello_world.sh` script en wijzig het zo dat het er zo uit komt te zien:

```
#!/bin/bash
# (c) 2020, Dennis Leeuw
# License: GPL v3
# Versie: 1.0
# Dit script print Hello world! Op het scherm

echo "Hello World!"
# Verlaat het script met error-code 0
exit 0
#END
```

Je hebt nu 5 regels commentaar toegevoegd. Het `#` symbool geeft dat het commentaar is en dat de interpreter (de shell) er niets mee moet doen. Als je het script opnieuw uitvoert zie je dat er niets gewijzigd is aan de uitvoer.

Vervang aan het commentaar het jaartal 2020 naar het jaartal waarin je nu leeft en verander de naam Dennis Leeuw in je eigen naam.

Het is een goede gewoonte om aan te geven wie een script gemaakt heeft, wat de licentie is en een korte beschrijving te geven van wat het script doet. Je werkt waarschijnlijk niet je leven lang op dezelfde afdeling of waarschijnlijk niet eens bij hetzelfde bedrijf. Beheerders na jou kunnen op deze manier snel zien wie iets gemaakt heeft en wat het script doet.

Ik laat mijn scripts meestal eindigen met `#END` zodat ik weet of het script compleet is. Als het script begint met een she-bang en eindigt met `#END` dan is het script compleet.

2.5 Variabelen

Het is soms handig om variabelen gebruiken:

```
$ aap=1
$ echo $aap
```

Zoals je ziet gebruiken we bij het toewijzen van een waarde aan een variabele (stop 1 in de variabele `aap`) geen `$`-teken voor de variabele. Alleen bij het gebruik van de variabele zetten we er een `$`-teken voor.

Variabelen in de shell hebben ook geen type, er bestaat geen integer of een string, dus we kunnen net zo makkelijk doen:

```
$ aap='Dag aap! '
$ echo $aap
```

Tot nog toe heb je kennis gemaakt met twee variabelen van de shell. `PATH` en `?`. Er zijn er nog veel meer die standaard beschikbaar zijn. Om er een paar te noemen:

```
$ echo $USER
$ echo $SHELL
$ echo $HOME
$ echo $PWD
```

om een complete lijst te krijgen van alle variabelen die in je huidige sessie tot je beschikking staan is er het commando `env`. Als je de waarde van een variabele wil wijzigen gebruik je `export`:

```
$ echo $PATH
$ PATH=".:${PATH}"
$ export PATH
$ echo $PATH
```

met deze opdracht hebben we de `.` directory toegevoegd aan de `PATH` variabele. Als we een commando aanroepen en het komt voor in de directory waar we op dat moment in staan dan zal het dat commando uitvoeren. We hoeven dan niet meer het hele pad of de `./` op te geven. Een variabele is een plek om data in op te slaan voor later gebruik. We kunnen ons script aanpassen op de volgende manier om een voorbeeld te geven van het gebruik van een variabele in scripts:

```
#!/bin/bash
# (c) 2020, Dennis Leeuw
# License: GPL v3
# Versie: 1.1
# Dit script print Hello world! Op het scherm

output="Hello world!"
echo $output
# Verlaat het script met error-code 0
exit 0
#END
```

We hebben nu een variabele met de naam `output` toegevoegd en deze variabele hebben we de waarde (data) `"Hello world!"` gegeven.

Bij het `echo`-commando gebruiken we de variabele om weer onze vertrouwde uitvoer te krijgen.

Let er op dat bij het declareren (het van een waarde voorzien) van de variabele er geen dollar-teken voor de variabele staat, maar bij het gebruik wel.

De shell heeft ook een aantal ingebouwde variabelen. Deze variabelen worden met allemaal hoofdletters geschreven. Om verwarring te voorkomen is het dus handig om je eigen variabelen niet met allemaal hoofdletters te schrijven. De meest gebruikte shell-variabelen in scripts zijn `$USER` die de gebruikersnaam bevat, `$HOME` die de home-directory bevat en `$PWD` die het complete directory pad bevat vanaf de root tot de directory waarin de gebruiker stond toen hij het script startte. Probeer dit script uit:

```
#!/bin/bash
# (c) 2020, Dennis Leeuw
# License: GPL v3
# Versie: 1.1
# Vertelt de gebruiker met welke gebruikersnaam hij of zij
#   ingelogd is, wat
#   zijn of haar home-directory is en in welke directory hij of zij
#   nu staat.

echo "Je bent ingelogd als $USER en je home-directory is $HOME"
echo "Je huidige directory is $PWD"
```

```
exit 0
#END
```

Variabelen worden dus gebruikt om data in op te slaan. Je kan ook de uitvoer van een commando in een variabele stoppen. Op Linux bestaat er het commando `date` dat op verschillende manieren informatie over de datum en tijd kan weergeven en waarmee je de datum en tijd op je computer kan zetten. Omdat `date` zoveel kan lijkt het in eerste instantie ingewikkeld, maar als je er een beetje ervaring mee op doet blijkt het een enorm handige tool te zijn. Tikken we alleen `date` in op de commandline en daarna enter, dan krijgen we van het systeem de huidige datum en tijd terug en in welke tijdzone we leven. Als we bijvoorbeeld alleen het jaar willen weten waarin we nu leven dan vragen we dat aan `date` door:

```
date +%Y
```

en alleen de maand is:

```
date +%m
```

dit is het maandnummer, willen de naam van de maand dan gebruiken we:

```
date +%B
```

Meer van deze opties vind je terug in de manual-pagina van `textttdate`. Lees deze door zodat je ook weet hoe je de dag, de uren en de minuten moet weergeven.

Om in een script de uitvoer van `date` in een variabele te zetten kunnen we het volgende gebruiken:

```
#!/bin/bash
# (c) 2021, Dennis Leeuw
# License: GPL v3
# Versie: 1.0
# Wat is de huidige datum en tijd

start="$(date '+%Y-%m-%d-%H-%M') "

echo ${start}

exit 0
#END
```

We hebben nu de uitvoer van het `date` commando aan een variabele gegeven en deze geprint. Let op de quotes! Als we parameters aan een commando willen voeren die ook een betekenis in de shell hebben, zoals in dit geval het

%-teken, dan moeten we de parameter tussen enkele quotes zetten zodat ze door de shell niet geïnterpreteerd worden.

De laatste manier om een variabele een waarde te geven is door aan de gebruiker input te vragen, hiervoor bestaat er het **read** commando. In een script zou je dat zo kunnen gebruiken:

```
#!/bin/bash
# (c) 2021, Dennis Leeuw
# License: GPL v3
# Versie: 1.0
# Vraag de gebruiker om zijn leeftijd

echo -n "Hallo $USER, hoe oud ben jij? "
read leeftijd

echo "De gebruiker $USER is $leeftijd jaar oud."

exit 0
#END
```

2.5.1 Opdracht: Werken met variabelen

1. Maak een script met twee variabelen. Je voornaam stop je in de eerste variabele en je achternaam in de andere.

Zorg ervoor dat het script op je scherm je voornaam en achternaam achter elkaar laat zien.

2. Schrijf een script dat aan een variabele het weeknummer geeft door gebruik te maken van het **date** commando. Normaal komt er een 2 cijferig getal uit, bij een cijfer onder de 10 wordt er een 0 voor het getal gezet. Schrijf het **date** commando zo dat er nooit een 0 voor gezet wordt.
3. Schrijf een script waarbij aan de variabele V1 de waarde 'Dit is een zin' wordt toegekend en maak een tweede variabele V2 met de waarde 'en die eindigt zo.'. Een derde variabele V3 moet nu de waarde van V1 en V2 bevatten en op het scherm worden afgebeeld.

2.6 if

Je kan een script gebruiken om alle commando's die je op de commandline gebruikt onder elkaar te zetten als een lange lijst. Beter is het als je ook controleert of een commando succesvol is verlopen. Eerder hebben we behandeld

dat de exit code van een commando terecht komt in de variabele `$?` . Als we een test zouden kunnen doen in een script om te zien of deze variabele 0 is, dus als alles goed is verlopen, dan kunnen we controleren of er commando's fout zijn gegaan.

Er is een commando dat `test` heet dat doet wat het zegt, je kan er dingen mee testen, en de uitkomst is waar (true) of nietwaar (false), andere opties zijn er niet. Dit commando gaan we niet direct in een script gebruiken, maar eerst eens op de commandline uitproberen om te zien wat we ermee kunnen.

```
$ touch test_bestand.txt
$ test -f test_bestand.txt
$ echo $?
$ test -f deze_bestaat_niet
$ echo $?
$ file -s test_bestand.txt
$ echo $?
```

Lees de manual-page van `test` eens door en zoek uit wat de `-s` optie betekent.

Het `test` commando kan ook gebruikt worden om waarden met elkaar te vergelijken. `test` maakt daarin in onderscheid tussen strings en integers (getallen).

```
$ test "Aap" = "Aap"; echo $?
$ test "Aap" = "Aapje"; echo $?
$ test "01" = "1"; echo $?
$ test "01" -eq "1"; echo $?
```

Zoals je gezien zult hebben test de `=` op het gelijk zijn van strings en `-eq` op het gelijk zijn van integers, waarbij 01 gelijk is aan 1.

Strings kan je vergelijken met `=` om te zien of ze aan elkaar gelijk zijn en met `!=` om te zien of ze niet aan elkaar gelijk zijn.

Met integers kan je nog veel meer vergelijkingen maken:

- eq** Controleer of integers gelijk zijn (equal)
- ne** Controleer of integers ongelijk zijn (not equal)
- gt** Controleer of eerste integer groter is dan de tweede (greater then)
- ge** Controleer of de eerste integer groter of gelijk is aan de tweede (greater or equal)
- lt** Controleer of eerste integer kleiner is dan de tweede (less then)
- le** Controleer of de eerste integer kleiner of gelijk is aan de tweede (less or equal)

In de manual pagina vind je nog veel meer mogelijkheden waarop je kan testen met **test**, maar voor ons zijn dit de belangrijkste opties. We gaan de opgedane kennis toepassen in shell-scripts.

Het eerste script dat we gaan maken is een script dat een **ping** doet naar een IP-adres en dat test of de ping goed verlopen is:

```
#!/bin/bash
# (c) 2021, Dennis Leeuw
# ping een host en kijk of dat succesvol is verlopen

ip="192.168.10.42"

ping -c1 ${ip} 2>/dev/null 1>/dev/null
if [ $? -ne 0 ]
    then
        echo "Het ${ip} is niet aanwezig op het netwerk"
    fi

exit 0
#END
```

We voeren hier het **ping** commando uit en sturen de error en andere berichten naar **/dev/null**. Als de return code in **\$?** niet 0 is, dus 1 of hoger is, dan heeft onze ping geen antwoord gekregen en vertellen we dat aan de gebruiker.

Het zal opgevallen zijn dat we bij de **if** twee bijzondere tekens hebben neergezet **[** en **]**. Deze tekens betekenen dat er een test gedaan moet worden. We hadden de regel ook zo kunnen schrijven:

```
if test $? -ne 0
```

Het is echter een goede gewoonte om in scripts de blokhaken, **[** en **]**, te gebruiken.

Stel dat we ook aan de gebruiker willen laten weten als het wel goed gegaan is. Dus in alle andere gevallen, dat kunnen we doen met **else**:

```
#!/bin/bash
# (c) 2021, Dennis Leeuw
# ping een host en kijk of dat succesvol is verlopen

ip="192.168.10.42"

ping -c1 ${ip} 2>/dev/null 1>/dev/null
if [ $? -ne 0 ]
    then
        echo "Het ${ip} is niet aanwezig op het netwerk"
    else
        echo "Het ${ip} is wel op het netwerk aangetroffen"
    fi
```

```
exit 0
#END
```

Probeer dit script ook eens met een ander IP adres, bijvoorbeeld 127.0.0.1.

De **if**, **then**, **else** kent in bash nog een toevoeging namelijk de **elif**. De **elif** is een samentrekking van **else** en **if** en kan gebruikt worden om meerdere tests na elkaar te doen. Bijvoorbeeld:

```
#!/bin/bash
# (c) 2021, Dennis Leeuw
# test een getal

getal=15

if [ ${getal} -le 10 ]; then
    echo "Getal is kleiner of gelijk aan 10"
elif [ ${getal} -ge 20 ]; then
    echo "Getal is groter of gelijk aan 20"
else
    echo "Getal ligt tussen 10 en 20"
fi
```

Neem dit script over en varieer de waarde van de variabele **getal**, zodat het script een keer door de **if** loopt, een keer door de **elif** en een keer door de **else**.

2.6.1 Opdracht: Werken met if

1. Maak een script dat aan de gebruiker een getal vraagt en geef weer of het getal kleiner of gelijk is aan 10, groter is dan 10 en kleiner of gelijk is aan 20, groter is dan 20 en kleiner of gelijk is aan 30 of groter is dan 30.
2. Maak een script dat test of een bestand leeg is. Als het bestand leeg is moet het verwijderd worden.
3. Maak een script dat aan de gebruiker om twee bestandsnamen vraagt. Test of de bestanden bestaan en kijk dan wel bestand nieuwer is. Laat aan de gebruiker weten welk bestand het nieuwste is. Tips: lees de manual-pagina van **test** en om te testen lees de manual-pagina van **touch**.

2.7 for

2.8 while

parameters 01 23

Alfabetische index

bash, [3](#)
builtin variables, [8](#)
commando
 bash, [3](#)
 cron, [4](#)
 date, [9](#)
 env, [7](#)
 exit, [5](#)
 export, [7](#)
 file, [5](#)
 ping, [12](#)
 read, [10](#)
 test, [11](#)
cron, [4](#)
date, [9](#)
echo, [8](#)
else, [12](#)
env, [7](#)
exit, [5](#)
export, [7](#)
file, [5](#)
for, [14](#)
hello_world.sh, [4](#), [6](#)
if, [10](#)
ingebouwde
 variabelen, [8](#)
magic number, [5](#)
ping, [12](#)
read, [10](#)
she-bang, [5](#)
shell, [3](#)
Shell scripting, [3](#)
 commentaar, [6](#)
 else, [12](#)
 for, [14](#)
 if, [10](#)
 ping, [12](#)
 variabelen, [7](#)
 while, [14](#)
shell-script, [3](#)
test, [11](#)
variabelen
 ingebouwde, [8](#)
while, [14](#)