Linux deel 2: CLI

D. Leeuw

28 februari 2021
v.0.2.0

Over dit Document

Dit document behandeld 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 bugfixes.

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 hoef 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.

Versienummer	Auteurs	Verspreiding	Wijzigingen
0.1.0	Dennis Leeuw	Wim Onrust	Initieel document
0.2.0	Dennis Leeuw	HEITO18IB-A	Toegevoegd: versie-nummering, de shell, begin van werken met bestanden

Tabel 1: Document wijzigingen

Inhoudsopgave

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 besturingssysteem 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

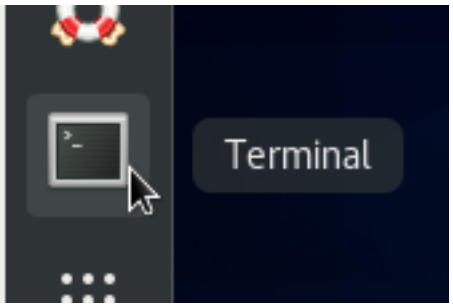
Waarom de commandline interface?

De eerste vraag is meestal waarom we de command line zouden gebruiken als we al een grafische interface hebben. Het meest simpele antwoord is dat Unix van oorsprong alleen maar een command line interface of CLI had. Maar een beter is antwoord is dat een grafische interface veel resources (geheugen en processor) gebruikt en die resources kunnen we beter inzetten voor de taken die we het systeem geven. Veel Linux machines draaien als servers in een serverruimte en staan op de achtergrond hun ding te doen, bijvoorbeeld als webserver. Voor die functie is geen grafische interface noodzakelijk terwijl op een drukbezochte website elk stukje processor of geheugen nodig kan zijn om de website soepel te laten lopen. Wat we niet nodig hebben installeren we dan ook niet op de machine, dus geen grafische interface.

Daarnaast zal je hopelijk ervaren dat omdat Linux op de schouders staat van de vele jaren ervaring uit de Unix wereld dat de command line een enorm krachtige interface is om mee te werken. Vaak kan je op de command line dingen sneller en makkelijker doen dan je in een grafische interface zou kunnen. Wees niet bevreest als dat in eerste instantie niet zo lijkt. De leercurve kan, zeker als je niet veel ervaring met bijvoorbeeld de command prompt of powershell van Windows hebt, soms erg stijl zijn.

2.1 Toegang tot de CLI

Een Linux server die geïnstalleerd is zonder grafische interface heeft op zijn monitor de prompt Login: daar kan je inloggen als gebruiker. Via de toets combinatie ALT en F1-F6 kan je schakelen naar verschillende consoles zodat je meerdere keren ingelogd kan zijn.

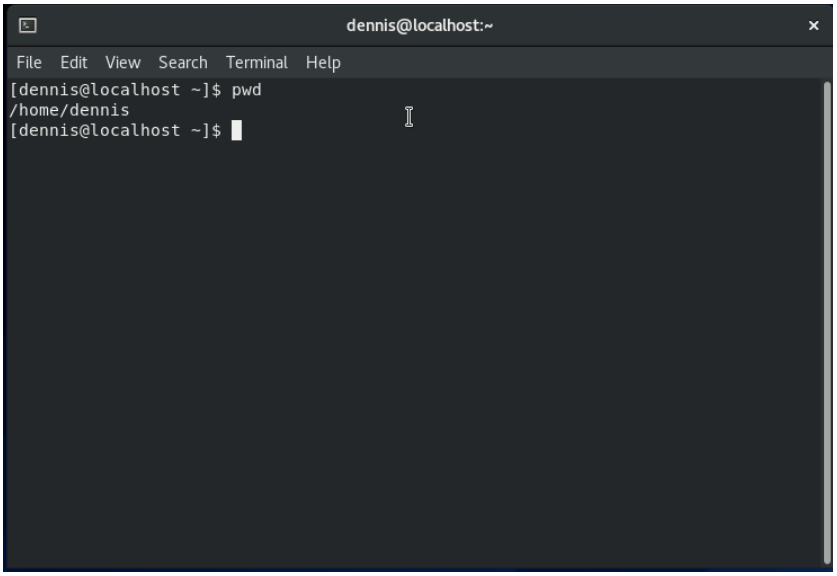


Figuur 2.1: Terminal op de Dash

Omdat wij werken vanuit een virtual machine is het werken met een console niet makkelijk en gebruiken we een terminal applicatie. Start Terminal of Terminal Emulator vanaf de Dash, zie figuur 2.1

2.2 De prompt

Als je op de command line bent heb je een commandprompt. De commandprompt is het deel voor de cursor, zie 2.2



Figuur 2.2: Console

Links staat de login naam, dennis, daarnaast op welke host je bent in gelogd, localhost, daarna volgt de directory waarin je je nu bevindt, ~, en tot slot de prompt voor een normale gebruiker: $. Als je ingelogt bent als

root dan staat er inplaats van een $ een #. In deze cursus zullen we alle commando's die je in moet of kan typen op de prompt vooraf laten gaan door een $ als je het commando als gewone gebruiker moet intypen en door een # als je het als root moet doen. De $ en # moet je dus niet intypen.

Typen we

```
$ pwd
```

dan zien we het pad naar de directory waarin we ons bevinden. In dit geval zal dat onze gebruikers directory zijn met als volledig pad /home/<gebruikersnaam>. Voor het voorbeeld waarbij de gebruiker dennis ingelogd is is dat dus /home/dennis. Merk op dat op Linux de directories gescheiden worden door de forward-slash, /, dit in tegenstelling tot Windows waar de backward-slash, \, gebruikt wordt als scheidingsteken.

Nu weten we waar we zijn. Als we willen weten onder welke naam we zijn ingelogd dan gebruiken we

```
$ whoami
```

Als je dit commando nu intypt dan zal je je eigen loginnaam zien.

```
$ hostname
```

laat zien op welke machine we zitten. Als ik dit commando intyp krijg ik terug 'localhost.localdomain', wat betekent dat mijn machine nog geen echte naam gekregen heeft. Dit kan op je eigen netwerk anders zijn, omdat dit afhankelijk is van server die de IP-adressen uitdeelt.

```
$ hostname -s
```

Geeft alleen de naam van de server terug. De -s staat voor short, ofwel kort.

Nu weet je een beetje wie je bent en waar je je bevindt, zowel op welke machine als in welke directory. Welkom thuis.

2.3 De shell

Alles wat je op de prompt intypt wordt opgevangen door de shell. De shell is een schil rond de kernel die commando's van gebruikers aanneemt en deze doorgeeft naar de kernel. Shell is het Engelse woord voor schelp en eem schelp zorgt ervoor dat het weekdier dat in de schelp leeft beschermt wordt tegen de buitenwereld. Dat is bij de shell net zo, de shell beschermt de kernel tegen de gebruiker. De shell wordt ook wel een command interpreter genoemt omdat hij commando's van de gebruiker interpreteert.

Linux gebruikt standaard de bash shell. Bijna elke linux distributie levert deze mee en heeft deze als standaard shell. Van oudsher werd op Unix systemen sh meegeleverd. Het was een van de eerste programma's die werd geschreven voor Unix door Ken Thompson. Tussen 1976 en 1979 schreef Stephen Bourne een vervanging voor sh wat de standaard werd in Version 7 Unix. De naam bleef echter sh op het Unix systeem. Toen het GNU project een vrij en open source systeem wilde maken was ook daar een van de eerste programma's die er moest komen een shell. Dat werd bash, de naam bash staat voor Bourne Again SHell. Het is een open source versie van de shell geschreven door Stephen Bourne.

Unix en Linux commando's en bestandsnamen zijn case sensitive. Dat betekent dat 'hello.txt' niet hetzelfde is als 'Hello.txt'. Dit zijn twee verschillende bestanden.

Commando's of opdrachten aan de shell hebben een vaste vorm (syntax). Ze zien er zo uit:

```
commando<spatie>optie(s)<spatie>argument(en)
```

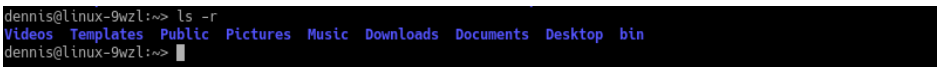
De spaties zorgen ervoor dat de shell weet wanneer een volgend deel begint, voor de eerste spatie staat het commando, daarna volgen er geen of enkele opties en tot slot zijn er geen of enkele argumenten. Bijna alle commando's houden deze syntax aan, hoewel er natuurlijk uitzonderingen zijn.

Het commando ls laat een lijst met bestanden zien. Als we het gebruiken zonder argumenten dan ziet dat er ongeveer uit als in figuur 2.3

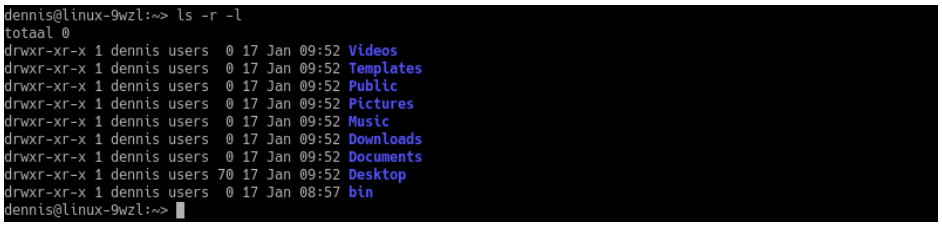
Figuur 2.3: Directory listing

Opties worden van argumenten onderscheiden doordat opties beginnen met een min-teken (-). Geven we aan ls een optie me, bijvoorbeeld -r voor reverse, of wel sorteer omgekeerd dan zien we wat we in figuur 2.3 zien. We zien nu dat Videos vooraan staat en bin achteraan.

We kunnen ook meerdere opties meegeven. De optie -l geeft een long list, ofwel een lijst die veel meer informatie per bestand of directory laat zien. Je



Figuur 2.4: Reverse order listing



Figuur 2.5: Reverse and long listing

mag de opties apart meegeven zoals we in figuur 2.3 gedaan hebben maar je mag ze ook samenvoegen zoals:

```
$ ls -lr
```

of

```
$ ls -rl
```

We kunnen ook een argument meegeven, bijvoorbeeld:

```
$ ls Documents/
```

Het argument is de naam van een directory, in dit geval Documents, en we krijgen geen output omdat de directory leeg is.

De linux shell heeft ook een handigheidje om het leven makkelijker te maken, of beter om minder te hoeven typen, dat handigheidje heet automatisch aanvullen. Als je op de prompt een p typt zonder een enter te geven en daarna de tab-toetst indrukt dan gebeurd er niets, druk je nog een keer op de tab-toets dan krijg je de melding:

```
Display all 177 possibilities (y or n)
```

Type n want we willen niet 177 mogelijkheden zien. Wat het systeem ons verteld heeft is dat het 177 commando's kent die met een p beginnen, voegen we nu aan onze p een w toe en typen we 1x tab. Dan vult de shell dit aan met de d en staat er pwd.

Dit automatische aanvullen kun je doen met commando's maar ook met bestanden en directories. Er wordt weleens gezegd dat je het toetsenbord van een Linux systeembeheerder kan herkennen aan de versleten tab-toets.

2.3.1 History

Een ander handigheidje van de shell is dat hij een geschiedenis (history) opslaat van gebruikte commando's. Met de pijltjes toetsen ↑ (pijltje-omhoog)

en ↓ (pijltje-omlaag) kan je door de geschiedenis van je commando's scrollen. Omhoog is terug in de tijd en omlaag is het omgekeerde. Met CTRL-c breek je af waar je gebleven bent en met enter voer je het commando uit.

Met CTRL-r kan je zoeken in je history. Type maar eens CTRL-r en dan de p. Er zal vrijwel direct pwd verschijnen. Op enter drukken is dan het enige dat nog nodig is om het commando uit te voeren. Als je toch besluit dat je het commando niet wil uitvoeren dan druk je op CTRL-c om de zoekfunctie te verlaten.

Met !! herhaal je het laatste commando dat je gedaan hebt en met !<commando> herhaal je het laatste <commando> dat je gedaan hebt. Type nu eens

```
$ !!
```

dan zal het pwd commando opnieuw uitgevoerd worden.

```
$ !hostname
```

zal de hostname -s uitvoeren want dat is het laatste hostname commando dat we gedaan hebben.

2.4 De home-directory

Wat waarschijnlijk wat uitleg behoeft is dat je prompt laat zien dat je je in de directory ~ bevindt. Dat is geen werkelijk bestaande directory maar een handige afkorting die we binnen de cli kunnen gebruiken om de home-directory van een gebruiker aan te duiden. Je eigen plek waar je je bestanden kan opslaan heet je home-directory en die kan dus ook aangeduid met de ~ (tilde). Je bevindt je dus in je eigen home-directory.

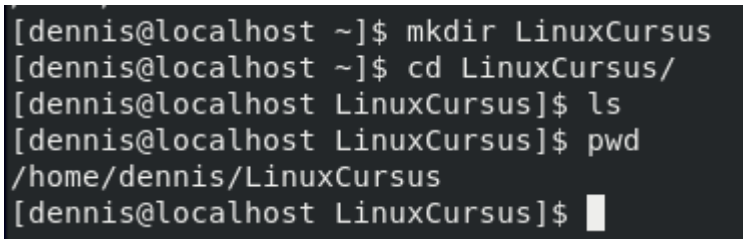
De werkelijke plek waar je je nu bevindt in het bestandssysteem kan je laten zien door

```
$ pwd
```

te typen. Het pwd commando toont de werk-directory en pwd staat dan ook voor print working directory. Je zult zien dat pwd iets terug geeft als

```
/home/dennis
```

waarbij 'dennis' vervangen is door je eigen gebruikersnaam. De / (forward slash) is het scheidingsteken dat in Linux gebruikt wordt om directory namen van elkaar te scheiden. Je bevindt je dus drie directories diep vanaf de root



Figuur 2.6: Het maken van de LinuxCursus directory

van het bestandssysteem. / is de root-directory, home/ is de tweede directory en dennis/ is de derde directory.

Laten we eens kijken wat er in onze directory te vinden is. Om een overzicht te krijgen van de directories en bestanden in onze directory typen we 'ls'. 'ls' is een afkorting voor list. Ofwel geef een lijst van aanwezige files en directories. Afhankelijk van de distributie of de systeembeheerder zullen er al wat zaken aanwezig zijn. We gaan er verder vanuit dat je een standaard CentOS installatie hebt gedaan zoals beschreven in een vorige hoofdstuk.

We gaan onze eerste eigen directory aanmaken. Type hiervoor:

```
$ mkdir LinuxCursus
```

met behulp van 'ls' kan je zien dat er een nieuwe directory bijgekomen is.

Met 'cd', change directory, kunnen we ons verplaatsen in de directory:

```
$ cd LinuxCursus
```

na de enter zal je zien dat de prompt gewijzigd is en een 'ls' geeft een lege output omdat er niets in de directory staat. Met 'pwd' kan je controleren dat je daadwerkelijk in de nieuwe directory staat.

Nu gaan we een aantal directories tegelijk aanmaken hiervoor typen we:

```
$ mkdir Data Documenten Muziek
```

na een 'ls' zie je nu drie nieuwe directories. Zo zie je dat je door meerdere argumenten mee te geven aan mkdir meer dan 1 directory kan aanmaken.

Met rmdir kan je directories weggooien.

```
$ rmdir Data
```

Dit gooit de Data directory weg, na 'ls' zie je nu nog maar twee directories.

Als we het systeem verder willen verkennen dan kunnen we met

```
$ cd \~{}
```

ervoor zorgen dat we weer in onze home-directory terecht komen, dat scheelt toch een hoop typen die ~. Als we nu

```
$ cd ..
```

typen dan gaan we terug in de directory boom. We komen in de /home/ directory terecht. De dubbele punten naast elkaar (..) staan voor de directory die 1 stap lager ligt in de boom. We staan nu in de home/ directory die standaard alle directories van alle gebruikers bevat, dus als er meer gebruikers op het systeem zouden zijn aangemaakt dan zouden ook van deze gebruikers de home-directories hier te vinden zijn. Een uitzondering is de home-directory van de systemadministrator van een Linux systeem. Deze beheerder heet 'root' en zijn of haar directory is /root/. Daar komen we later nog een keer op terug.

Als we weer

```
$ cd ..
```

typen dan komen we in de / directory terecht. Lager kunnen we niet gaat op een Linux systeem, we zijn nu bij de root-directory aangekomen. Let op de verwarring die kan ontstaan tussen de /root/ directory en de / directory beiden heten de root-directory maar de ene is van de gebruiker root en de ander is van het bestandssysteem.

Met alleen het 'cd' commando

```
$ cd
```

kom je ook weer terug in je eigen home-directory.

2.5 Shell variabelen

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.

2.5.1 Eigen 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 variable (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
```

Hoofdstuk 3

Het bestandssysteem

3.1 Commando's

Het nu volgende hoofdstuk word je uitgelegd waar de commando's op een Linux systeem staan, hoe je met de PATH variabele het zoeken naar commando's kan beïnvloeden en hoe een commando je laat weten of het goed is gegaan of dat er een fout is opgetreden. Ook krijg je in dit hoofdstuk te lezen hoe je als root commando's uit kan voeren.

3.2 Waar zijn de commando's?

Toen we eerder de p typten en daarna twee keer op de tab-toets toen zagen we dat we 177 commando's hadden die met een p begonnen. 177 commando's alleen al met een p dat is veel. Een gemiddeld Linux systeem bevat heel veel commando's en dat is omdat er in de Unix-wereld twee filosofieën zijn die bij elkaar aansluiten de eerste is het KISS-principle. KISS staat voor Keep It Simple, Stupid en is oorspronkelijk afkomstig uit de US Navy. En de tweede is Small is Beautiful en die is afkomstig uit de Economie[1]

Op Linux systemen kom je vele kleine commando's tegen die één ding goed doen. Dit heeft een aantal voordelen. Omdat ze maar één ding doen is de code simpel en is dus makkelijker te controleren op fouten. Omdat niet iedereen in elk programma weer dezelfde code hoeft te herhalen maar gebruik kan maken van iemand anders zijn programma is de totale hoeveelheid code klein, een compleet Linux systeem met webserver kan zonder grafische interface geïnstalleerd worden op een 8G USB-stick. De laatste reden is dat

[1]Small Is Beautiful: A Study of Economics As If People Mattered door E. F. Schumacher

wij als gebruikers vaak zonder te programmeren al heel complexe dingen met Linux kunnen doen omdat er al zoveel tools al beschikbaar zijn.

Het nadeel van heel veel kleine programma's is dat je het overzicht snel kwijt kan raken. Een van de simpele programma's is 'ls', dat is een afkorting voor list. Veel commando's in Linux zijn afkortingen. De oorspronkelijke ontwikkeling van Unix werd gedaan op systemen met toetsenborden die niet zo ergonomisch zijn als de onze. Ze hadden toetsen die je met enige kracht moest indrukken en na een dag programmeren hadden de programmeurs Ken Thompson en Dennis Ritchie en hun team vaak zere knokkels door overbelasting. Door commando's zo kort mogelijke namen te geven verminderden ze het aantal toetsaanslagen. Vandaar de korte commando namen.

Type

```
$ ls
```

en je zal een aantal blauwe directories op je scherm zien verschijnen. Als je nu

```
$ ls /usr/bin/
```

typt dan verschijnen er allemaal groene commando's op je scherm, of beter ze scrollen van je scherm in twee kolommen, omdat het er te veel zijn. Het past niet op je scherm. Als we de hele lijst willen zien dan zullen we gebruik moeten maken van een programma de uitvoer van 'ls' opdeelt in pagina's die zoveel regels bevatten dat ze het scherm vullen. Een programma dat dat doet heet 'more'. De kunst is nu om de uitvoer van 'ls' te koppelen aan 'more' en daarvoor is er de pipe (|) of de pijp. Het pipe-character koppelt twee commando's aan elkaar:

```
$ ls /usr/bin/ | more
```

met de spatie-balk kan je nu pagina voor pagina bekijken en met de letter q verlaat je 'more'. Nu is more wel heel simpel en kan het alleen dat wat je nu gezien hebt. Makkelijker zou het zijn als je omhoog en omlaag door de commando's kan gaan, en misschien zelfs wel zou kunnen zoeken in zo'n lange lijst. Dat kan ook, daarvoor hebben we de opvolger van 'more' die meer kan en 'less' heet want less is more.

```
$ ls /usr/bin/ | less
```

Nu kan je met de spatie-balk door de pagina's gaan, met de pijltjes omhoog en omlaag per regel door de lijst gaan, met PgUp en PgDn per pagina omhoog en omlaag gaan en met / kan je zoeken. Type maar eens als je in 'less' zit

```
/firefox
```

Zo kom je bij het commando 'firefox' uit. Ook hier weer is de q-toets de manier om 'less' te verlaten.

We hebben nu gezien dat heel veel commando's terug te vinden zijn in de /usr/bin/ directory. Maar dit is maar één plek waar commando's te vinden zijn. Commando's vind je terug in de (s)bin/ directories. We gebruiken hier een meervoud omdat ze zich op verschillende plekken kunnen vinden. Je bent in de / directory al de bin en sbin directories tegen gekomen. sbin is voor de systeembeheerder commando's en bin voor commando's die ook door de normale gebruiker gebruikt kunnen worden.

Maar een Linux systeem zit iets complexer in elkaar. Je hebt waarschijnlijk ook al de /usr/ directory gezien. In die directory kom je een vergelijkbare structuur tegen als in /. Vaak wordt er gezegd dat commando's in /bin/ en /sbin/ nodig zijn om het systeem op te starten. Alles ze niet direct nodig zijn voor het opstarten dan vind je ze in /usr/bin/ en /usr/sbin/. Dat geldt ook voor de libraries in /lib/ en /usr/lib/.

Linux is een open source systeem, dus een systeembeheer kan ook bepalen dat bepaalde software door hemzelf gecompileerd wordt omdat hij bijvoorbeeld een nieuwere versie beschikbaar wil maken dan door de distributie geleverd wordt. Deze lokaal gecompileerde software wordt dan geïnstalleerd in /usr/local/. Dus /usr/local/bin bevat commando's die afkomstig zijn van lokaal gecompileerde software en /usr/local/lib/ de libraries.

En als laatste is er nog de /opt/ directory. /opt/ is voor voorgecompileerde software die niet onderdeel is van de distributie maar die bijvoorbeeld gecompileerd is door een commerciële leverancier van software.

Om te bepalen welke directories gebruikt worden voor het zoeken naar een commando is er een variabele aanwezig in de shell waarin we werken en die variabele heeft de logische naam PATH. Type maar eens:

```
$ echo $PATH
```

Dit levert een output op die er ongeveer zo uit ziet: /usr/local/bin:/usr/bin:/bin. Voor een gebruiker met dit PATH wordt er voor een commando eerst gezocht in /usr/local/bin, daarna in /usr/bin en als laatste in /bin/.

Met het commando su kan je een andere gebruiker worden (switch user). Type eens:

```
$ su -
```

geef het root wachtwoord en type

```
# echo $PATH
# exit
```

Na het su commando moet je het password van de root gebruiker geven dat je ingesteld hebt tijdens de installatie. Je zal nu zien dat het PATH van de root gebruiker ook de sbin/ directories bevat. De root gebruiker heeft dus veel meer commando's tot zijn beschikking dan een normale gebruiker.

3.3 Error codes

Stel dat we een commando hebben uitgevoerd en we krijgen niets terug, hoe weten we dan zeker dat het goed gegaan is? Dat weten we omdat een commando ook een exit-code terug geeft. Een exit-code van 0 betekend dat alles goed gegaan is en alles boven 0 dat er iets fout gegaan is. De man-page van het programma kan je vaak meer vertellen over welke exit-code wat betekent als je er niet uit komt met de beschrijving van de error. Type het volgende

```
$ cat hello.txt
$ echo $?
$ cat Hello.txt
$ echo $?
```

na de eerste echo $? krijg je een 0 en na de tweede echo $? een 1. Dat komt omdat het eerste commando uitgevoerd kan worden en het tweede niet. Het bestand Hello.txt (met een hoofdletter) bestaat niet.

Het $-teken betekent dat we te maken hebben met een variabele. Het vraagteken is een speciale variabele die de shell gereserveerd heeft om de exit-code in op te slaan.

3.4 FHS - Filesystem Hierarchy Standard

We staan nog steeds in de / directory als we daar ls typen dan zien we allemaal verschillende directories op ons scherm verschijnen.

Net als met de standaardisatie van Unix in een POSIX standaard werden er in het begin op Linux Distributies soms bestanden in verschillende directories neergezet. Dat is voor programma's die op die systemen moeten draaien niet handig. Als de ene distributie /var/db heeft voor het plaatsen van databases en de ander /var/databases dan schept dat verwarring. De oplossing die hiervoor gekomen is is de Filesystem Hierarchy Standard. Deze is beschikbaar op https://refspecs.linuxfoundation.org/fhs.shtml. Hier gaan we heel globaal in op een aantal belangrijke directories, mocht je alle ins en outs willen weten dan raden we je aan om het document een keer te lezen.

De basis van het bestandssysteem wordt bepaald door de / directory, ook de root genoemd omdat de vertakkende directories op een boom structuur lijkt en het Engelse root betekend stam.

Een ls van / laat ons een aantal verschillende directories zien. Laten we beginnen met /boot/. Deze directory bevat bestanden die cruciaal zijn voor het opstarten maar die geen commando zijn. Hier vinden we de kernel en bestanden die behoren bij de bootloader.

De /dev/ directory bevat de namen van beschikbare devices. Devices worden worden besproken in het hoofdstuk over devices. Dus daar gaan we later nog op in.

De /etc/ directory bevat de configuratiebestanden van het systeem. Als je een instelling wil wijzigen is dit de plek om te gaan zoeken.

/home/ bevat de directories waarin gebruikers hun bestanden kunnen zetten. Een uitzondering hierop is de directory waarin de root gebruiker (de baas of administrator van het systeem), zijn bestanden kan opslaan. Die directory is /root/.

/var/ is de directory voor de systeem opslag van variabele data zoals bijvoorbeeld de logbestanden die je dan ook kan vinden in /var/log/.

/srv/ bevat de data van de diensten die door het systeem wordt aangeboden. Data van web- of ftp-servers kan hier gevonden worden.

Hoofdstuk 4

Linux documentatie

Bijna elk linux-systeem is standaard voorzien van een uitgebreide set van documentatie. De meeste documentatie bevat de man-pages. 'man' is een afkorting voor Manual ofwel handleiding. Dat zal je vaker tegen gaan komen dat commando's op een Linux en andere Unix-systemen afkortingen zijn. Het afkorten scheelt typen, wat cruciaal was op de oude stugge toetsenborden van de PDP-systemen waarop Unix is ontworpen. Op deze toetsenborden kreeg je zere vingers van het typen, dus alles dat typen bespaarde was meegenomen.

4.1 man-pages

De op het systeem aanwezig help van een Unix-achtig systeem is te vinden in de man-pages. De online Manual is verdeeld in hoofdstukken.

1. Executable programs or shell commands
2. System calls (functions provided by the kernel)
3. Library calls (functions within program libraries)
4. Special files (usually found in /dev)
5. File formats and conventions eg /etc/passwd
6. Games
7. Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
8. System administration commands (usually only for root)

9. Kernel routines [Non standard]

Voor gebruikers zijn de belangrijkste hoofdstukken die over de commando's gaan, 1 en 8, en die over de configuratie bestanden 5. Om een beetje vertrouwd te raken met de man-pages gaan we de manual page over man bekijken. Type:

```
$ man man
```

Met de pijltjes toetsten omhoog en naar beneden kan je door de pagina scrollen en met q verlaat je de manual-pagina. Scrollend door de pagina kom je ook de hierboven al genoemde hoofdstuk indeling tegen. Verder kom je kopjes tegen met hoofdletter. Veel voorkomende koppen zijn:

```
NAME: naam van het commando met een korte uitleg

SYNOPSIS: syntax van het commando

DESCRIPTION: een beschrijving van het commando

OPTIONS: welke opties kent het commando

EXAMPLES: voorbeelden hoe het commando te gebruiken

AUTHORS: wie heeft de manual-page geschreven

SEE ALSO: doorverwijzingen naar andere documentatie
```

Om snel te zoeken naar bepaalde stukken in een manual-page kan je de / gebruiken. Als je een man-page open hebt staan en je typt

```
/EXAMPLES
```

en deze kop bestaat in de pagina dan kom je meteen bij de examples terecht. Snel door de man-pages scrollen kan door gebruik te maken van PgUp en PgDn toetsen.

4.2 Waar vind ik iets?

Om uit te vinden welk commando je kan gebruiken om iets op het systeem te bereiken kan je man -k gebruiken. Een alternatief commando is apropos. Type eens:

Figuur 4.1: Het zoeken van informatie

```
$ apropos 'make directories'
```

je vindt dan het 'mkdir' commando. Het nadeel van dit zoek systeem is dat het redelijk specifiek is.

```
$ apropos 'make directory'
```

doet niets. Als je het dus niet meteen vindt probeer dan enkelvoud- en meervoudsvormen.

Heb je een commando gevonden waarvan je denkt dat het is wat je nodig hebt probeer dan man -f of whatis.

```
$ whatis mkdir
```

Dit geeft een korte beschrijving van een commando en voor de volledige manual gebruiken we het man commando:

```
$ man mkdir
```

Extra uitleg van een commando kan vaak ook gevonden worden door -h of –help aan het commando toe te voegen met een spatie.

```
dennis@sticky10vm:~$ mkdir -h
mkdir: invalid option -- 'h'
Try 'mkdir --help' for more information.
dennis@sticky10vm:~$ mkdir --help
Usage: mkdir [OPTION]... DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short options too.
  -m, --mode=MODE   set file mode (as in chmod), not a=rwx - umask
  -p, --parents     no error if existing, make parent directories as needed
  -v, --verbose     print a message for each created directory
  -Z                   set SELinux security context of each created directory
                         to the default type
      --context[=CTX]  like -Z, or if CTX is specified then set the SELinux
                         or SMACK security context to CTX
      --help     display this help and exit
      --version  output version information and exit

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation at: <https://www.gnu.org/software/coreutils/mkdir>
or available locally via: info '(coreutils) mkdir invocation'
```

Figuur 4.2: Gebruik van -h of –help

Zoals je ziet in 4.2 dat -h een error melding geeft en ons vertelt dat we --help moeten gebruiken. Ook wordt er verteld wat we de volledige documentatie online kunnen vinden in de info-documentatie.

4.3 Info

The GNU-project heeft een eigen documentatie systeem ontworpen dat info genoemd wordt. Het is een hypertext, dus met links, gerelateerd systeem. Dit is dus documentatie dat je naast man tegen komt op systemen. Mocht je dus in man niet vinden wat je zoekt, misschien kan je dan eens info proberen. Ook in info werken de pijltjes voor het scrollen en is q er weer om het systeem te verlaten. Een extra functie is de enter-toets die je kan gebruiken om een link te volgen.

4.4 /usr/share/doc

Een andere plek waar we op ons systeem informatie kunnen vinden is in de /usr/share/doc directory. Met het commando ‘ls’ kan je een listing van een directory opvragen. Type

```
$ ls /usr/share/doc
```

en een enorme lijst van directories vliegt over je scherm. Dit is de documentatie van elk pakket dat op het systeem geïnstalleerd is. Een linux systeem is opgebouwd uit allerlei software pakketten die vanaf source code door de distributie maker gecompileerd zijn. De informatie die met de source

code meekwam, zoals de licentie-bestanden, eventuele FAQ-bestanden, README's etc. die vind je terug in de subdirectories van /usr/share/doc. Belangrijk zijn vaak de voorbeeld configuratie-bestanden.

4.5 Internet

Als laatste documentatie bron willen we graag het Internet vermelden. Omdat GNU/Linux een open source besturingssysteem is is er heel veel online te vinden. De kans dat jij als eerste tegen een probleem aanloopt is zeer klein. Het is dus naast de voornoemde bronnen een van de eerste zaken om te raadplegen.

De broncode van alle software is openbaar en veel van de projecten hebben dan ook hun eigen website met vaak ook een eigen forum om contact te houden met gebruikers. Veel informatie is dan ook terug te vinden in de fora en de FAQs.

Hoofdstuk 5

Gebruikers, groepen en rechten

Het Unix besturingssysteem en dus ook Linux is altijd bedoeld geweest als multi-user systeem. Het was dus de bedoeling om meer dan één gebruiker op een systeem te laten werken. Bij het ontwerp is er dan ook rekening mee gehouden dat er rechten moesten zijn voor verschillende gebruikers en ook is er van het begin af aan rekening gehouden met dat mensen in groepen zouden moeten kunnen samenwerken.

Dit hoofdstuk gaat over deze gebruikers en groepen en vertelt hoe je de rechten per gebruiker en groep kan zetten.

5.1 Gebruikers en groepen

Bij het inloggen heb je een gebruikersnaam en wachtwoord opgegeven en bij de installatie heb je ook een wachtwoord moeten opgeven voor de gebruiker root. Op het systeem zijn dus minimaal al twee gebruikers aanwezig. Op een Linux systeem kunnen ook processen een gebruiker hebben. Dus een proces kan onder een bepaalde gebruiker werken zodat andere gebruikers niet bij dit proces kunnen. Processen zijn taken die op de achtergrond draaien zoals bijvoorbeeld een webserver.

De database met gebruikersnamen is een bestand dat staat in de /etc directory. Het bestand heet passwd en dat kan je bekijken met less.

```
less /etc/passwd
```

5.2 Werken als root

De root-gebruiker is op een Linux-systeem almachtig. Deze gebruiker mag alles inclusief het systeem stuk maken en dat is helemaal niet zo moeilijk om te doen. Juist omdat root alles mag is het niet verstandig om als de root gebruiker op een systeem te werken. Doe zoveel mogelijk als een normale gebruiker. Pas als het echt niet anders kan doe je het als root.

Inloggen als root zou je eigenlijk nooit moeten doen. Als je iets als root wil doen gebruik je sudo (Super User Do).

Om te werken als een andere gebruiker is er su (switch user), dit kan je natuurlijk ook gebruiken om root te worden en die verleiding is waarschijnlijk groot. Maar wen jezelf aan om dat niet te doen en sudo te gebruiken om handelingen als root uit te voeren.

5.2.1 sudo

Met het sudo (super user do) commando kan je commando's uitvoeren alsof ze van root zijn, je moet daar dan natuurlijk wel de rechten voor hebben niet iedereen mag dat doen. Wie er rechten heeft om wat te doen wordt bepaald door het /etc/sudoers bestand of door bestanden in de /etc/sudoers.d directory.

Om te zien wat er in de home-directory van root staat kan je het volgende commando geven:

```
$ sudo ls /root
```

5.2.2 su

Met het commando su (switch user) kan je een werken als een andere gebruiker, mits je het wachtwoord weet van die gebruiker.

Om ook alle omgevingsvariabelen van die gebruiker mee te krijgen moet je aan su het min-teken(-) meegeven. Dat ziet er dan zo uit:

```
$ su - mies
```

5.3 Toegangsrechten op bestanden en directories

5.3.1 Bestandstypen

5.3.2 read, write and execute

5.3.3 Het 4de-bit

5.4 User Personal Group

Hoofdstuk 6

Werken met bestanden

6.1 Everything is a file

6.1.1 Bestandstypen

6.2 Directories

6.3 Bestanden maken en stdin, stdout en stderr

Zorg dat je in de directory LinuxCursus staat en type:

```
$ touch hello.txt
```

Na de Enter lijkt er helemaal niets te gebeuren. Dit is met de meeste Linux commando's het geval. Als het goed gegaan is dan laten ze niets weten, een beetje als "geen nieuws, is goed nieuws". Doen we een 'ls' dan zien we dat er een bestand is aangemaakt dat hello.txt heet.

Met touch kunnen we dus bestanden aanmaken, dit zijn lege bestanden. Type maar eens:

```
$ cat hello.txt
```

dan zal je zien dat er weer niets op je scherm verschijnt. En dat is goed! Het 'cat' commando plaatst de inhoud van een bestand op het scherm en daar we een leeg bestand hebben opgevraagd is wat er op het scherm komt dus niets en omdat dat succesvol is verlopen hoeft cat ook geen foutmelding te laten zien en met de wetenschap dat geen nieuws, goed nieuws is is cat klaar.

In de acties die we nu hebben doorlopen staat er dat echo tekst naar het scherm schrijft en cat bestanden op het scherm afbeeldt. Dat is inderdaad wat er gebeurt, maar vanuit het Linux gezien niet helemaal correct. Zowel echo als cat schrijven naar de standaard output en in de terminal is het scherm de standaard output. De standaard output wordt vaak afgekort als stdout.

We kunnen de standaard output ook omleiden (redirect) naar bijvoorbeeld een bestand:

```
$ echo 'Ik werk met Linux' > hello.txt
```

We zien nu dat de zin die we met echo afbeelden niet meer op het scherm verschijnt. Hij is verdwenen en er lijkt weer helemaal niets gebeurd te zijn. Als we nu

```
$ cat hello.txt
```

doen dan zien we waar onze zin is gebleven. Hij is hello.txt terecht gekomen. We hebben de stdout van echo in hello.txt gestopt.

Laten we dat nog eens doen:

```
$ echo 'Hello World!' > hello.txt
```

Doen we een cat van hello.txt dan zien we dat onze eerste zin verdwenen is en er alleen nog 'Hello World!' in hello.txt zit. We hebben kennelijk ons bestand overschreven met nieuwe inhoud. We kunnen ook tekst toevoegen aan een bestand:

```
$ cat 'Ik werk met Linux' >> hello.txt
```

door gebruik te maken van het dubbele groter dan teken voegen we een regel toe aan het eind van het bestand. De oude regel zie je met cat als eerste en daaronder komt onze nieuwe regel.

Zou er als we een stdout hebben ook een standaard input zijn en kunnen we daar dan van lezen? Ja, die is er. Als je typt:

```
$ cat < hello.txt
```

dan vertellen we eigenlijk dat cat de invoer (stdin) op het scherm moet afbeelden. Maar ja, dan moeten we een spatie en een <-teken extra typen en dat doen we liever niet.

Naast de standaard input en standaard output is er ook nog standaard error (stderr), waar de foutmeldingen naartoe gaan. Laten we eens een fout maken door een niet bestaan bestand aan cat te geven:

```
$ cat Hello.txt
```

Je krijgt nu een foutmelding dat Hello.txt niet bestaat. Linux is case-sensitive dus hello.txt is niet hetzelfde als Hello.txt. De standaard output, input en error zijn genummerd in Linux. Stdin is 0, stdout is 1 en stderr is 2. Nu we dit weten zouden we het volgende kunnen doen:

```
$ cat Hello.txt 2> Hello_error.txt
```

We zien nu geen foutmelding meer op ons scherm en hebben de stderr omgeleid (redirect) naar het bestand Hello_error.txt. Doen we nu een

```
$ cat Hello_error.txt
```

dan zien we dat de foutmelding daar is opgeslagen.

Deze vormen van het redirecten (omleiden) van data stromen gebeurt in Linux heel vaak. Programma's schrijven bijvoorbeeld de fouten die ze tegen komen naar een log bestand. Dit doen ze door de stderr te redirecten en de error regels toe te voegen aan het bestand, bijvoorbeeld 2»error.log. Als ze dit doen met een datum- en tijdmelding dan kan je heel makkelijk problemen opzoeken.

Wij hebben nu geleerd om bestanden aan te maken en om invoer en uitvoer te redirecten.

6.4 Het gebruik van een editor

Op de commandline heb je geen menu's en vaak geen muis om door een applicatie te navigeren. Het maken van documenten is dan ook een stuk lastiger dan in een grafische interface. Toch zijn er oplossingen gevonden om op de commandline te werken met bestanden. Een tekstverwerker op de commandline heet een editor. Er zijn verschillende editors bedacht en in gebruik. Een van de oudste voor Unix geschreven editors is **vi**.

Een van de grote namen achter Unix is Ken Thompson. De eerste drie commando's die hij schreef voor het jonge Unix systeem waren **as** (assembler), **ed** (editor) en **sh** (shell). Dennis M. Ritchie bracht verbeteringen aan op ed en vanaf 1969 tot 1976 bleef dit de editor op een Unix systeem. In 1976 kwam Billy Joy en Chuck Haley met een nieuwe editor die ex werd genoemd.

Voor ex schreef Billy Joy ook een soort interface om er makkelijker mee te kunnen werken en die wrapper om ex noemde hij vi (visual interface). Vanaf 1979 werd ex geintergreerd in vi en was er alleen nog vi. Later werd vi onderdeel van de Single Unix Specification en daarmee een editor die op bijna elk Unix systeem aanwezig is en dat is nogsteeds het geval. Op bijna alle beschikbare Unix systemen, van BSD tot Linux en Mac OS X is vi aanwezig. Dat is dan ook het voordeel van het aanleren van het werken met vi dat de kennis op verschillende platformen gebruikt kan worden.

6.4.1 vi, pico, nano

De eerste redelijk gebruiksvriendelijke editor op Unix was vi. De vi editor kent twee modi. De eerste modus is de edit mode en de tweede is de command mode. Standaard start vi op in de command mode waarin je commando's kunt geven om bestanden te laden of op te slaan en waarin je functies als knippen en plakken kan uitvoeren. De edit modus is die waarin je je tekst invoert. Dit onderscheid maakt voor beginnende gebruikers vi soms verwarrend.

Naast vi zijn er ook andere editors voor Unix-achtige systemen ontwikkeld. De meeste bekende zijn pico en nano. Pico was de oorspronkelijke editor. Nano is ontwikkled door het GNU-project en is een vervanging van pico omdat pico een licentie had die "problematisch"was. Dat probleem is inmiddels opgelost, maar nano biedt zoveel extra mogelijkheden dat velen de voorkeur geven aan nano.

Het grote voordeel van nano ten opzichte van vi is zijn gebruiksvriendelijke interface. Nano kent geen edit en command mode zoals vi. Nano gebruikt control codes om commando's te geven en is direct beschikbaar voor de invoer van tekst van de gebruiker.

6.4.2 vim

De naam vim staat voor Vi IMporved. Of wel een verbeterde versie van vi. Bram Molenaar een Nederlandse software ontwikkelaar schrijft als sinds 1991 aan de code van vim en zijn verbeteringen zijn zo populair dat op alle Linux systemen alleen nog vim geïnstalleerd wordt. Je kan op een Linux systeem vim opstarten als vi waarmee je zoveel mogelijk de functionaliteiten krijgt als het oude vi en je kan vim opstarten als vim waarmee je alle nieuwe toevoegingen van Bram Molenaar en zijn medeontwikkelaars krijgt.

vim is niet een van de makkelijkste editors maar heeft als grote voordeel, zoals eerder genoemd, dat het beschikbaar is op elk willekeurig Unix systeem.

vim opstarten

De meeste gebruikelijke manier om vim op te starten is door aan vim meteen een bestandsnaam mee te geven:

```
$ vim bestand.txt
```

Als je klaar bent met het toevoegen van tekst kan je met **:wq** afsluiten. Dit slaat het document op (w) en sluit af(q). Wil je de editor verlaten zonder de gemaakte wijzigingen op te slaan, dan gebruikt je **:q!** om dat te doen. Het doet een quit (q) zonder verdere vragen stellen.

Een andere manier om vim op te starten is door geen bestandsnaam mee te geven:

```
$ vim
```

de editor weet nu niet onder welke bestandsnaam een bestand opgeslagen moet worden. Bij de write (w) moet je nu dus de bestandsnaam meegeven: **:w bestand.txt** slaat het bestand dat je gemaakt hebt op als bestand.txt. Na deze opdracht kan je met **:q** vim afsluiten. Om tekst toe tevoegen of te wijzigigen in vi moet je vanuit de command mode naar de edit mode gaan. Hiervoor zijn verschillende commando's beschikbaar. De meest gebruikte zijn **i** van insert of **a** van add. Om de edit mode te verlaten gebruik je de ESC toets.

Met het gebruik van het **i** commando voeg je tekst in vóór de plek van de cursor. Door gebruik te maken van **a** voeg je tekst in na de positie van de cursor. Vanuit de command modus kan je het character waarop je staat verwijderen door gebruik te maken van het **x** commando, eigenlijk is dit het knippen commando, maar kan gebruikt worden om snel wat characters te verwijderen. Het **D** commando verwijdert de text vanaf de plek waarop je staat tot het einde van de regel. Het einde van de regel is tot de plek waar je ENTER gegeven hebt, dus niet tot het einde van de regel op het scherm.

Het verwijderen van een complete regel van begin tot eind doe je met **dd**. Dit commando kan je uitbreiden met aantallen regels door tussen de d's een getal op te geven. Bijvoorbeeld **d2d** verwijdert 2 regels vanaf de plek waar je staat. En zo kan je ook met **dl** letters verwijderen. De combinatie **d5l** verwijdert vanaf de positie van de cursor 5 characters. Om hele woorden weg te gooien gebruik je **dw** en ook daar geldt de mogelijkheid om meerdere woorden in één keer weg te gooiden met **d3w** gooi je 3 woorden achter elkaar weg. Met het **x** commando kan je één enkel character knippen. Het commando is gelijk aan **dl**.

De traditionele manier om een selectie en kopie van een stuk tekst te maken is het gebruik van het **y** commando. Het **yl** commando selecteert characters, **yw** selecteert woorden en **yy** selecteert regels.

Een speciale functie van vim en niet van vi is het gebruik van **v** om een visuele selectie maken, met de pijltjes toetsen kan je nu bepalen hoe groot de selectie worden moet. Het commando **v** geeft je de mogelijkheid om de selectie op character niveau te maken. Met **V** maak je selecties per regel, hier gebruik je de omhoog en omlaag pijltjes toetsen om je selectie groter of kleiner te maken.

Het **p** commando kan gebruikt worden om text te plakken. Het **p** commando is plakken achter de positie van de cursor en **P** is plakken voor de positie van de cursor. Met de pijltjes toetsen kun je in de edit mode bewegen door een bestand. De pijltjes links en rechts bewegen de cursor een character per keer naar links of rechts, de op en neer pijltjes bewegen de cursor een regel op en neer.

In de command mode kan je met ^ bewegen naar het begin van een regel en met **$** naar het einde van de regel. Een grotere sprong is met **gg** naar het begin van een document en met **G** naar het einde van het document.

Met **w** spring je een woord vooruit.

Je kan ook springen naar een bepaalde regel. Hiervoor gebruik je bijvoorbeeld :10. Dit commando sprint naar regel 10 van boven.

Met zoek opdrachten kan je ook door een document bewegen. Het / teken geeft het begin van een zoekopdracht aan. Dit werkt alleen in de commandmodus. Met /**zoeken** zoek je naar het woord zoeken in de tekst. Met de letter **n** kan je naar het volgende zoekresultaat springen en met **N** naar het voorgaande.

Hoofdstuk 7

Zoeken en vinden

In dit hoofdstuk ga je leren hoe je op een Linux systeem naar bestanden kan zoeken. Ook ga je leren hoe je in bestanden kan zoeken naar patronen, waarbij woorden patronen kunnen zijn, maar er kan veel meer. Dat alles komt in dit hoofdstuk aan bod.

Om de opdrachten uit dit hoofdstuk te kunnen maken is het van belang om de volgende commando's uit te voeren:

```
$ cd $HOME
$ mkdir Zoeken_en_vinden
$ cd Zoeken_en_vinden
$ touch plaatje.jpg
$ touch Plaatje.png
$ touch Slaatje.txt
$ mkdir -p Muziek/Rammstein/
$ touch Muziek/Rammstein/mutter.mp3
$ mkdir -p dit/is/een/heel/diepe/directory
$ mkdir -p dit/is/ook/een/heel/diepe/directory
```

7.1 Globbing

Globbing is het gebruiken van wildcards. Wildcards zijn bijvoorbeeld de asterisk (*) en het vraagteken (?). Een asterisk staat voor geen of meer characters en een vraagteken staat voor één character. Zo kunnen we ontbrekende characters invullen of aanvullen als we niet meer helemaal zeker weten hoe een bestand heet. Stel dat we weten dat we een document gemaakt hebben met de naam `Plaatje` maar dat we niet meer weten of dat een jpg, png of gif plaatje is. Dan kunnen we een overzicht van alle bestanden in een directory opvragen met:

```
$ ls Plaatje.*
```

We krijgen dan de bestanden met een willekeurige extensie terug.

Globbing is een functie die door de shell wordt uitgevoerd. De shell vervangt dus eerst de wildcards door wat er op het bestandssysteem staat en voert dat uiteindelijk aan `ls`. Als je quotes om het argument zet dam gaat `ls` opzoek naar een bestand dat `Plaatje.*` heet. Er verschijnt dan keurig de melding dat dat bestand niet bestaat.

Zo kunnen we ook zoeken op de bestandsnamen waarvan we niet meer zeker weten of we het met een hoofdletter of kleine letter hebben geschreven door gebruikt te maken van het vraagteken:

```
$ ls ?laatje.*
```

Het toeval wil dat we ooit het recept van een vriend hebben opgeschreven over hoe we huzarensalade moeten maken dus kregen we van het laatste commando terug:

```
plaatje.jpg  Plaatje.png Slaatje.txt
```

Als we alleen de bestandsnamen die echt het plaatje bevatten willen vinden dan zullen we gebruik moeten maken van een ander optie die de shell ons biedt, namelijk de blokhaken ([]). In shell-globbing betekenen de blokhaken een reeks. Een reeks kan zijn [1234567890], of [abcdefg], wat trouwens simpeler geschreven kan worden als [0-9] en [a-g], maar een reeks mag ook zo simpel zijn als [pP], dus de hoofdletter P en de kleine letter p.

```
$ ls [Pp]laatje.*
```

en zo krijgen we precies terug wat we willen.

Reeksen in globbing zijn heel handig. Je kan bijvoorbeeld een reeks maken [a-zA-Z0-9] zodat je 3 reeksen combineert en zo alles hebt dat geen leesteken bevat. Als je ook bestandsnamen hebt die een spatie kan bevatten maak je er [\ a-zA-Z-0-9] van. Let op het escape (\) character voor de spatie. Zonder dat character zou de shell aan ls twee argumenten meegeven, want er staat een spatie en spaties (white-space) scheiden argumenten. Dus ls krijgt de opdracht om een list te doen van '[' en van 'a-zA-Z0-9]' en dat is niet wat we willen. We willen dat onze reeks één argument is, vandaar dat we de spatie "escapen" zoals dat heet. Door het escape-character behandeld de shell de spatie niet als scheidingsteken van argumenten maar als onderdeel van de reeks.

7.2 Zoeken naar bestand

Om op de commandline bestanden te zoeken die ergens op het filesysteem staan is er het commando `find`. De syntax van het `find` commando ziet er

ongeveer zo uit:

```
find <path> <options> <action>
```

`find` zoekt vanaf het opgegeven `path` naar bestanden die voldoen aan de opgegeven opties en voert daar de opgegeven `action` op uit. De standaard actie is om de naam van het bestand inclusief het complete pad te printen naar de standaard output. De zoekactie van `find` is als je dat niet limiteert recursief en dus door alle subdirectories. Dat betekent ook dat als je / als pad op geeft dat het hele bestandssysteem doorzocht wordt (dat kan even duren). Met zo'n grote boom van directories is het van belang dat je op een simpele manier bestanden terug kan vinden. Om te zoeken naar bestanden of directories is er het commando 'find'. De syntax van 'find' ziet er zo uit:

```
$ find \ [-H] \ [-L] [-P] [-D debugopts] [-Olevel] [starting-
    point...] [expression]
```

Dit is geknipt en geplakt uit de man-page waar je uitgebreide documentatie kan vinden over find. Om een idee te krijgen hoe find werkt type:

```
$ find \~{} -name ``Muziek'' -print
```

Bij mij op het systeem was het antwoord

```
/home/dennis/LinuxCursus/Muziek
```

Bij jou is dennis natuurlijk weer vervangen door je eigen gebruikersnaam. Print is de standaard functie van find, en daarom vonden de programmeurs van find dat als je geen opdracht meegeeft dat find dan ook print doet. Dus korter kan het zo

```
$ find \~{} -name ``Muziek''
```

Die ~ is makkelijk als je in je home-directory wil zoeken, maar wat als je in bijvoorbeeld /etc/ staat en in die directory wil zoeken? Daar is ook over nagedacht. We hebben al '..' gezien als een aanduiding voor een lager gelegen directory, maar zo is er ook de '.' als we 'deze'-directory bedoel. En met deze bedoelen we de directory waarin we nu staan. Dus we kunnen ook het volgende doen:

```
$ find . -name ``Muziek''
```

En voor wie niets tegen typen heeft mag je natuurlijk ook de hele directory opgeven

```
$ find /home/dennis/ -name ``Muziek''
```

Met -name geven we op dat we naar een bestandsnaam zoeken en een bestandsnaam kan ook een directory zijn. Als we onderscheidt willen maken tussen bestanden en directories kan kunnen we een -type meegeven:

```
$ find . -type d -name ``Muziek''
```

Zal dezelfde output geven want type d is een directory, maar als doen

```
$ find . -type f -name ``Muziek''
```

dan vinden we niets, want het type f is een regulier bestand en Muziek is een directory.

Om dat te testen doen we het volgende

```
$ touch \~{}/LinuxCursus/Documenten/leeg\_bestand.txt
$ find . -type f -name ``leeg\_bestand.txt''
```

Met touch kan je een nieuw leeg bestand aanmaken en dat hebben we gedaan en daarna hebben we find naar dat nieuwe bestand laten zoeken. Nou lijkt dit een wat onzinnige actie omdat we al weten waar het bestand is, maar het geeft ons een optie om een andere functie van find te demonstreren, namelijk het zoeken naar lege bestanden op het systeem:

```
$ find . -size 0
```

de -size optie geldt alleen voor bestanden, de -empty optie laat naast lege bestanden ook lege directories zien

```
$ find . -empty
```

Het find commando kent nog veel meer opties zoals zoeken naar de datum en tijd waarop een bestand is aangemaakt of een moment later of eerder dan een bepaalde datum en tijd. De man-page van find documenteert al deze verschillende opties en op Internet is heel veel uitleg te vinden hoe je find met al deze opties kan gebruiken. Een laatste functie van find willen je nog meegeven. De kan behalve met -print find ook andere dingen laten doen dan de gevonden elementen printen, je kan find ook vertellen om een actie uit te voeren, zoals het deleten van de gevonden elementen:

```
$ find . -size 0 -delete
```

een ls van LinuxCursus/Documenten/ zal laten zien dat het bestand leeg_bestand.txt niet meer bestaat. Pas wel op, dit kan gevaarlijke situaties opleveren.

```
$ find LinuxCursus/ -empty -delete
```

Dit commando zoekt in de LinuxCursus naar alle directories en bestanden die leeg zijn. Bestanden hadden we niet meer, maar er stonden nog wel twee directories in (Documenten en Muziek), beide directories waren leeg en werden door find dan ook keurig verwijderd van het systeem. Toen kwam find nog de directory LinuxCursus tegen, en ja die was inmiddels ook leeg, dus heeft find die ook weggegooid!

7.3 Zoeken in bestanden

7.4 Regular Expressions

Hoofdstuk 8

Mounten van bestandssystemen

Wat je zal zijn opgevallen als je een Windows-gebruiker bent is dat Linux geen drive letters heeft.

8.0.1 Mounten van disks

Een disk of partitie op een disk wordt gemount op het bestandssysteem. Om dit duidelijk te maken moeten we wat dieper in het systeem duiken en een paar dingen doen die we pas later in dit boek uitleggen. Dus type nauwkeurig het volgende commando over:

```
$ mount | grep ' / '
```

Voor en achter de / staat een enkele spatie. Dit geeft iets terug als:

```
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro)
```

Wat dit betekent is dat de eerste partitie op de harddisk die het systeem kent als sda gekoppeld (gemount) is op /. USB-disks kunnen op dezelfde manier gekoppeld worden aan het systeem. De directory die daarvoor gereserveerd is is /media/. In de /media/ directory zijn er directories per gebruiker waar gebruikers hun USB-disks kunnen mounten. Later zullen we hier uitgebreider op terug komen.

/mnt is de directorie die van oudsher aanwezig is om lokaal media op te mounten en zou nu alleen nog door de beheerder gebruikt moeten worden voor het tijdelijk mounten van disks.

Hoofdstuk 9

Systeem inventarisatie

Hoofdstuk 10

Software installeren

Hoofdstuk 11

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.

11.1 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.

11.2 Hello World - een eerste script

We gaan een bestand maken met de naam hello_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.

11.3 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
```

11.4 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 gebeurd 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.

11.5 Variabelen

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 gebuikersnaam 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 huidge 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 date. 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 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
```

11.6 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.

```
#/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 [ $? != 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.

Stel nu 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 [ $? != 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.

FIXME: test [] en elif

11.7 for

11.8 while

parameters 01 23

Hoofdstuk 12

Programmeertalen

Van oudsher werden Unix systemen veel gebruikt door programmeurs er zijn dan ook veel talen ontwikkeld en overgezet naar Linux. Natuur is er een C-compiler. De meest gebruikte is die uit het GNU project die de GNU Compiler Collection (GCC) heet omdat hij naar C ook compilers bevat voor C++, Objective-C, Fortran, Ada, Go en D.

Ook voor scripting talen zijn er veel interpreters aanwezig zoals voor PHP, Perl, Python en Java.

Daarnaast is er via verschillende kanalen nog veel meer te installeren.

Index