Linux de Command Line Interface

D. Leeuw

25 januari 2023
v.0.3.0

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 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: versienummering, de shell, begin van werken met bestanden
0.3.0	Dennis Leeuw		Toegevoegd: Werken met bestanden, Everything is a file, Bestandtypen

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) en dient als voorbereiding op het MBO ICT Systems and Devices Expert examen. Voor het leren gebruiken van de grafische interface en de command line maken we gebruik van CentOS en om kennis te maken met het gebruik van Linux als server installeren we Debian. De keuze om CentOS als werkstation te installeren en Debian als server is volledig willekeurig. Het doel is dat de studenten kennis maken met de rpm en dpkg package managers en leren dat het ene Linux systeem het andere niet is.

Alle Linux systemen zullen geïnstalleerd worden als virtuele machines op Virtual Box (https://www.virtualbox.org/). Door gebruik te maken van virtuele machines zijn we niet afhankelijk van de onderliggende hardware. De keuze voor VirtualBox heeft te maken met het feit dat dit product gratis te gebruiken is en beschikbaar is voor zowel Windows, Mac OS X als Linux.

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

Hoofdstuk 2

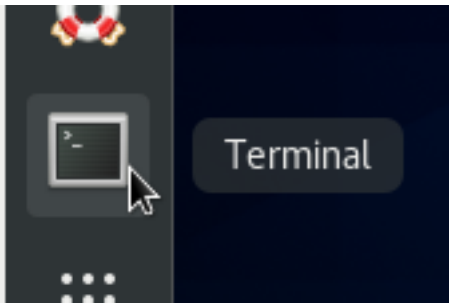
Waarom de commandline interface?

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 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 hebt met bijvoorbeeld de command prompt of powershell van Windows, 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.

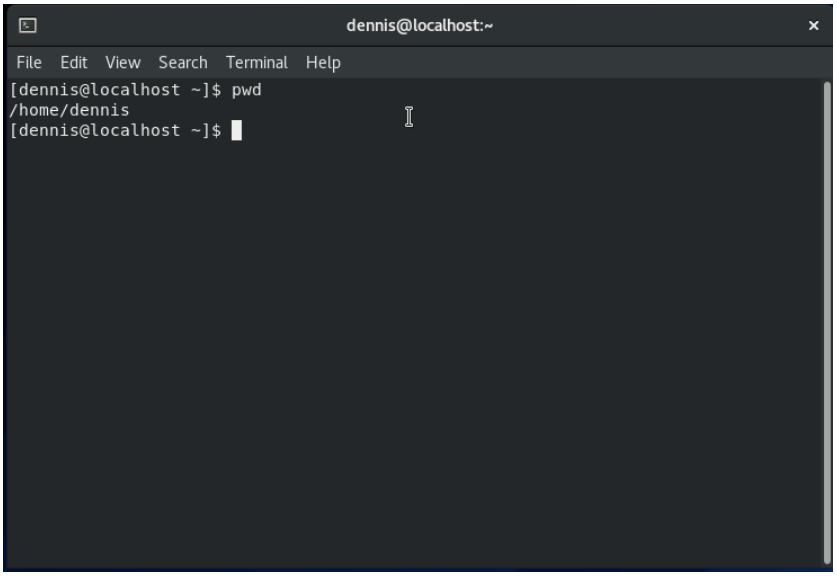


Figuur 2.1: Terminal op de Dash

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 hostname 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 een 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 ook 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

```
dennis@linux-9wzl:~> ls
bin  Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
dennis@linux-9wzl:~>
```

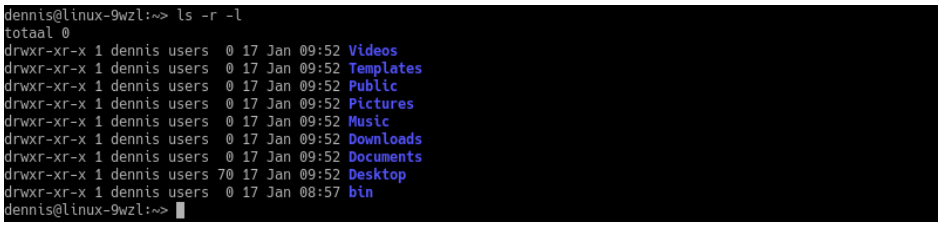
Figuur 2.3: Directory listing

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

```
dennis@linux-9wzl:~> ls -r
Videos  Templates  Public  Pictures  Music  Downloads  Documents  Desktop  bin
dennis@linux-9wzl:~>
```

Figuur 2.4: Reverse order listing



Figuur 2.5: Reverse and long listing

```
$ 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-beheerder 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 worden 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

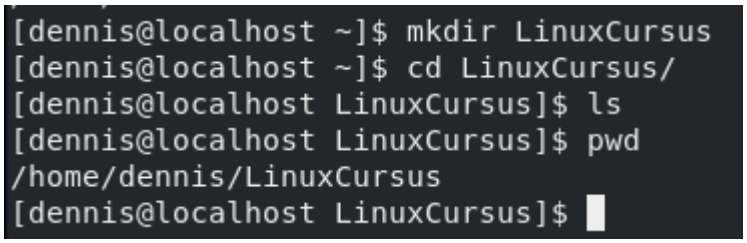
```
$ pwd
```

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 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



Figuur 2.6: Het maken van de LinuxCursus directory

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 Everything is a file

In Unix en Unix-like operating systems zoals Linux is het basis principe dat alles een bestand (file) is. Dit betekent dat alles binnen het systeem; documenten, directories, harddisks, printers, toetsenborden, maar ook processen weergegeven worden als bestanden. Het voordeel hiervan is dat dezelfde commando's en API's gebruikt kunnen worden voor verschillende onderdelen van het besturingssysteem.

Het filesysteem is een enkele boom van bestanden en directories, zonder onderscheid tussen disks en partities. Zelfs verplaatsbare media zoals USB-sticks en DVDs zijn onderdeel van deze boom als ze 'gemount' zijn. Ook processen (/proc) en de kernel (/sys) is voor een groot deel benaderbaar via het bestandssysteem.

Met `ls` kan je in bijvoorbeeld `/proc` kijken en zien welke processen er zijn. Je ziet er nummer staan en die nummers komen overeen met de process nummers die ook `ps` heeft. Doe maar eens:

```
$ ps aux
```

en je ziet in de tweede kolom dezelfde nummers staan. `ps` is dan ook het commando om te zien welke processen er op je systeem actief zijn. We zullen ps in een volgend hoofdstuk nog uitgebreid behandelen.

3.1.1 Bestandstypen

In de eerste kolom van `ls -l` vinden we de bestandrechten en het geeft tevens aan met welk type bestand we te maken hebben. Naast normale bestanden (zoals tekstbestanden) hebben we op POSIX-compliant systemen ook speciale bestanden zoals directories.

Bestandstype	Beschrijving
normaal bestand	Documenten, etc.
directory	Directories bevatten geen bestand, maar een overzicht van de bestandsn
Symbolic link	een link naar een bestand die over bestandssystemen heen kan gaan
Block device	Een apparaat waar van of waar naar toe data in een random manier ge
Character device	Een apparaat waar data in een stroom van characters naar of naar toe
FIFO	Ook bekend als named pipes. Een pipe verbindt het ene proces met het
Socket	Verbindt net als FIFO's processen, maar dan op een manier dat er twee

3.2 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

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.

4.1 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 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

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

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/.

Als we willen weten waar een commando vandaan komt, dan kunnen we `which` gebruiken:

```
$ which ls
```

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.

4.2 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.

Hoofdstuk 5

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.

5.1 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.

5.2 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 5.1: 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 5.2: Gebruik van -h of –help

Zoals je ziet in 5.2 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.

5.3 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.

5.4 /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.

5.5 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 6

Werken met bestanden

Als alles een bestand is in Linux dan is het werken met bestanden het belangrijkste wat er is. Dit hoofdstuk gaat je dan ook de basis beginselen bij brengen van het werken met bestanden.

6.1 Directories

Om data op een computer te structureren is het handig om de data te verdelen over directories. Directories zijn ook bekend als mappen en folders. Wij zullen alleen nog spreken van directories omdat binnen de Unix-wereld de meest gebruikte term is.

Een directory maak je aan met het commando `mkdir`:

```
$ mkdir LinuxCursus
```

Met `ls` kan je controleren of de directory ook daadwerkelijk aangemaakt is.

Je kan ook meerdere directories tegelijk aanmaken door ze als een lijst op te geven, gescheiden door spaties:

```
$ mkdir Aap Noot Mies
```

Soms wil je ook een heel pad gelijk aanmaken met:

```
$ mkdir Boom/Roos/Vis/Vuur
```

gaat dat niet lukken, want de Boom directory bestaat niet. Gelukkig kan je aan `mkdir` en optie meegeven die vertelt dat `mkdir` ook alle onderliggende directories moet aanmaken:

```
$ mkdir -p Boom/Roos/Vis/Vuur
```

Gebruik `ls` om te controleren dat alle directories aanwezig zijn.

Tot slot wil je ook instaat zijn om directories weg te gooien. Met `rmdir` kan dir als de directories leeg zijn.

```
$ rmdir Aap Noot Mies
```

gooit keurig alle aangemaakte directies weg. Controleer dit met `ls`. Maar doen we:

```
$ rmdir Boom
```

dan krijgen we een error melding, want de `Boom` directory is niet leeg. We zullen dus eerst alle andere directories moeten weggooien, te beginnen met `Vuur`, dan `Vis`, dan `Roos` en tot slot kunnen we pas `Boom` weggooien. Gebruik `rmdir` om alle directories **behalve** Boom te verwijderen. De `Boom` directory moet dus blijven bestaan.

6.2 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.

6.3 Bestanden kopieëren, verplaatsen, hernoemen, verwijderen

Om bestanden te kopieëren gebruiken `cp` van het Engelse copy:

```
$ cp hello.txt Boom/hello.txt
```

We kunnen ook gelijk de naam veranderen als we dat willen:

```
$ cp hello.txt Boom/Hallo.txt
```

Om bestanden verplaatsen gebruiken `mv` van het Engelse move. Het verschil met copy is dat een bestand niet meer op de oorspronkelijke plek terug te vinden is. Bij move heb je dus maar 1 bestand na de handeling, na copy heb je 2 bestanden.

```
$ mv Boom/Hallo.txt .
```

Het `mv` commando kunnen we ook gebruiken om bestanden van naam te veranderen:

```
$ mv Hallo.txt hallo.txt
```

Veranderd de naam.

Voor het weggooien van bestanden gebruiken we `rm` van remove.

```
$ rm hallo.txt
```

Omdat alles een bestand is op een Linux systeem zijn ook directories bestanden, speciale bestanden, maar toch bestanden. We hebben al gezien dat we met `rmdir` lege directories weg kunnen gooien. Zouden we nu `rm` kunnen gebruiken om ook directories weg te gooien. Ja, dat kan, maar ook hier geldt dat de directory leeg moet zijn.

```
$ rm Boom
```

geeft weer een foutmelding. Het systeem zegt tegen ons dat `Boom` een directory is. Als we tegen `rm` vertellen dat hij de zaken recursive weg met gooien, dan zal de hele boomstructuur wegggegooid worden:

```
$ rm −r Boom
```

Hoofdstuk 7

Het gebruik van een editor

Dit is geen onderdeel van het LPI Linux Essentials examen, maar voor het gebruik van Linux op het MBO leek het mij een goed idee om enige basis vaardigheden te hebben in het gebruik van `vi` *als editor. Ook voor de rest van dit boek wordt enige kennis van* `vi` *veronderstelt.*

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.

7.1 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.

7.2 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.

7.2.1 vim opstarten

Het kan zijn dat `vim` nog niet geinstalleerd is. Mocht dat het geval zijn, installeer dan `vim` via de packagemanager voor jouw systeem.

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 8

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
```

```
$ cd ~
$ mkdir ZIB
$ echo "Een commando op Unix systemen is vaak kleine en simpel en doet 1 ding
    goed." > ZIB/kleineCommandos.txt
$ echo "Door verschillende van deze tools te combineren kunnen complexe taken
    volbracht worden." > ZIB/complexeTaken.txt
$ echo "De documenten voor dit boek houden hetzelfde principe aan." > ZIB/boek.txt
$ echo "Het zijn korte stukken tekst die gezamenlijk een compleet boek vormen. We
    noemen dit het small-is-beautiful-principe." > ZIB/small_is_beautiful.txt
$ echo "We kunnen dit op boeken, tekst en software code toepassen." >
    ZIB/toepassen.txt
```

8.1 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 dan 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.

8.2 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!

8.3 Zoeken in bestanden

Soms zou je willen dat je in een bestand kunt zoeken. Natuurlijk kan je met in een tekstverwerker of een editor zoeken in een bestand. Maar wat nu als je niet zeker meer weet in welk bestand het was dat je iets geschreven had. Dat kan zomaar gebeuren als je een boek zoals dit aan het schrijven bent. Dit boek is opgebouwd uit allemaal kleine bestanden die te samen het boek vormen. Op deze manier hou ik de onderwerpen gescheiden en hoef ik niet elke keer te scrollen om bij een ander deel te komen. Ik kan ook twee onderwerpen in twee verschillende terminals te gelijk open hebben staan en zo parallel aan elkaar werken. Dit deel gaat over hoe we in bestanden kunnen zoeken zonder dat we een tekstverwerker open hebben staan met ons document erin.

Er zijn verschillende commando's die we kunnen gebruiken, de meest gebruikte is denk ik `grep`. Met `grep` kan door regular expressions te gebruiken zoeken in bestanden. Meer over regular expressions vind je in de volgende sectie. Nu gaan we vooral kijken naar hoe het `grep` commando werkt.

`grep` is de Global Regular Expression Parser. De naam `grep` vindt zijn oorsprong in een zoekopdracht uit `ed`, een editor: `g/re/p` wat stond voor zoek door de hele tekst (**g**lobal) naar deze **r**egular **e**xpression en druk deze af (**p**rint). `re` werd dan vervangen door een regular expression.

Om te zien hoe `grep` werkt gaan we eerst een paar bestanden met inhoud aanmaken.

```
$ cd ~
$ mkdir ZIB
$ echo "Een commando op Unix systemen is vaak kleine en simpel en doet 1 ding
    goed." > ZIB/kleineCommandos.txt
```

```
$ echo "Door verschillende van deze tools te combineren kunnen complexe taken
    volbracht worden." > ZIB/complexeTaken.txt
$ echo "De documenten voor dit boek houden hetzelfde principe aan." > ZIB/boek.txt
$ echo "Het zijn korte stukken tekst die gezamenlijk een compleet boek vormen. We
    noemen dit het small-is-beautiful-principe." > ZIB/small_is_beautiful.txt
$ echo "We kunnen dit op boeken, tekst en software code toepassen." >
    ZIB/toepassen.txt
```

8.4 Regular Expressions

Regular expressions zijn speciale karakters waarmee je makkelijker in complexe data-sets kan zoeken. De naam regular expression wordt vaak afgekort tot regexp of regex.

Regular expressions zoeken regels waarin een bepaalde een bepaalde reeks aan karakters voorkomt en beeldt dan de gehele regel af. Je kan zoeken in bestanden of in de output van een commando.

De basis van regular expressions is een aantal karakters met een speciale betekenis:

.	Is een willekeurig karakter
?	Is exact 1 karakter
*	Herhaal de voorgaande expressie 0 of meer keren
^	Begin van de regel
$	Einde van de regel
\	Zorg ervoor dat speciale karakters als echte karakters behandeld worden
{}	Zorg dat dat een regular expression een eenheid (groep) vormt

Deze speciale karakters kunnen we gebruiken om regular expressions te maken. We beginnen met de . en het ?:

```
$ mkdir regex
$ cd regex
$ touch aap.txt
$ touch ap.txt
$ touch pa
$ touch filetxt
$ ls | grep a?*p
$ ls | grep a.*p
```

merk op dat het verschil in output van de twee `ls` commando's zit in het aantal keren dat een letter voor moet komen. ?* zegt dat er 1 of meer karacters moeten zijn, terwijl .* zegt dat het 0 of meer karakters moet zijn.

```
$ ls | grep $\textasciicircum$a
$ ls | grep a$
```

Met het ^ zoeken we vanaf het begin van de regel, dus de regel moet beginnen met een a. Met de $ zoeken we vanaf het einde, dus de regel moet eindigen met een a.

Als we willen bestanden willen zien die op .txt eindigen dan willen dat de punt onderdeel is van onze zoek opdracht en willen we niet dat de punt gezien wordt als een willekeurig karakter. Vergelijk de volgende twee commando's en hun uitkomst:

```
$ ls | grep '.txt'
$ ls | grep '\.txt'
```

Let op de ticks om de regular expression. De ticks zorgen ervoor dat de regular expression bij `grep` terecht komt en niet al door de shell wordt uitgevoerd. De shell is natuurlijk de eerste die de complete commando regel krijgt. Het is de shell die moet beslissen wie welk deel uitvoert. Voor `ls` is dat niet zo moeilijk daar wordt de opdracht aan `ls` overgelaten, maar voor de regular expression bij `grep` ontstaan er twee mogelijkheden. De shell lost zelf al een deel van de regular expression op en geeft wat er over blijft aan grep of de shell geeft de complete regular expression aan grep. Dat laatste is wat we in dit geval willen. Vergelijk met je hiervoor gekregen uitkomsten eens met:

```
$ ls | grep \.txt
```

Wat hopelijk opvalt is dat ondanks de \ er niet gefilterd wordt op .txt maar op alles dat een willekeurig karakter heeft met daarachter txt. Dat komt omdat de shell de \. al matched met de output van `ls`. Dit is dus wel iets om op te letten met regular expressions. Het is altijd veilig om de regex tussen ticks te zetten. Dan weet je zeker dat de regex bij grep uitkomt en niet bij de shell.

Soms willen we kunnen aangeven hoe vaak een bepaald karakter achter elkaar voorkomt. Daarvoor hebben we de curly braces (accolades).

```
$ ls | grep -E 'a{2}'
```

Zoek zelf in de man-page van grep op wat -E betekent.

8.5 Zoek en vervang

We kunnen ook regels zoeken in een bestand die aan een bepaald patroon (regex) voldoen en dan het patroon vervangen door iets anders. Een van de meest gebruikte tools daarvoor heet `sed`.

```
$ ls -1
$ ls | sed -e 's/aap/noot/'
```

overal waar aap stond staat nu noot. Op de disk is er niets gewijzigd, we hebben alleen de output van ls aangepast.

Om iets meer van `sed` te leren gaan we eerst een bestand aanmaken:

Me stage heeft plaatsgevonden **in** 2018. De reden dat ik voor dit bedrijf gekozen heb is omdat het een overheidsinstantie is en graag wil leren hoe is om **in** zo'n bedrijf te werken, verder wil ik me ontwikkelen doormiddel het uitvoeren van wat ik heb geleerd op me school.

Me werkzaamheden was het helpen van mensen met hun probleem binnen en buiten het bedrijf, ik zat op de Servicedesk.

Wij krijgen onze technologische aparaten waaronder laptops computers als er iets stuk en niet zomaar te repareren valt. We vragen eerst of er nog garantie op de product is, zo ja sturen we de produkten naar op naar de leverancier. Zo niet dan kijk we er zelf naar.

Ik had een werkende computer die hele tijd vast liep en rare geluide maakte. Ik heb zelf een stappenplan gemaakt om te controleren of de computer hardware of software problemen had.

1. Zit de stroom goed op de voeding?
2. Zijn alle componenten goed aangesloten
3. Is de product schoon (stofvrij)
4. Handige Informatie opschrijven van de BIOS

Uiteindelijk was de fout dat er teveel stof in de roosters zat van de computer, stof kan computers slecht laten draaien en de lucht cirkulatie verminderen van binnen. Waardoor ze kunnen vastlopen.

We gebruiken Topdesk. Dit is een melding registratie systeem. Hierin worden incidenten genoteerd en is er een stukje communicatie tussen de melder en de oplosser. De melder of een van de servicedesk medewerkers maken de melding. Daarna kijkt de oplosser naar de melding en stuurt een mail naar de des betreffende persoon. De oplosser belt vaak ook naar de melder om de probleem op te lossen en meld de melding af na afloop. De melder ontvangt een mail dat de melding is afgehandeld.

Hier zitten wat taal fouten in. Die gaan we zoeken en ook vervangen. We beginnen met het gebruik van Me aan het begin van de eerste twee alinea's. Dat moet natuurlijk Mijn zij.

```
$ sed –e 's/^Me/Mijn/'
```

Op het scherm zien we nu de verbeterde tekst, op disk staat echter nog de oude tekst. We kunnen de verbeterde tekst natuurlijk met het groter dan teken wegschrijven naar disk met een nieuwe bestandsnaam. Makkelijker is om de verbeterde tekst in het al bestaande bestand aan te passen:

```
$ sed –ie 's/^Me/Mijn/'
```

We zien nu geen output, omdat de tekst op de disk gewijzigd is. Gebruik `cat` om te zien dat de tekst inderdaad gewijzigd is. Zoek in de man-page op wat de betekenis is van de -i en de -e opties.

Een ander 'me' die fout is is de 'me' voor 'me school' maar de 'me' voor 'me ontwikkelen' is bijvoorbeeld goed. We moeten dus een regular expression schrijven ide alleen matched op 'me school':

```
$ sed –ie 's/me\ school.$/mijn\ school/'
```

Let op de \ voor de spatie. Een spatie is in de shell een scheidingsteken en dat willen we nu niet. We bedoelen een echte spatie, dus moeten we hem escapen.

Hoofdstuk 9

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.

9.1 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
```

De wachtwoorden staan in een ander bestand, dat heet `shadow`. Dit bestand kan je met `less` niet bekijken, omdat alleen de beheerder (root) hier rechten voor heeft. De wachtwoorden zijn niet leesbaar, maar geencrypt, opgeslagen. Met het `passwd`-commando kan je je wachtwoord wijzigen.

Gebruik `grep` om je eigen gegevens uit `/etc/passwd` te halen:

```
$ grep dennis /etc/passwd
```

vervang hierbij *dennis* door je eigen gebruikersnaam.

De output van de vorige commando zal er ongeveer zo uit zien:

```
dennis:x:1000:1000:Dennis Leeuw,,,:/home/dennis:/bin/bash
```

Het is een soort database waarin de verschillende elementen gescheiden zijn door een :.

1. gebruikersnaam (login-naam)
2. werd vroeger gebruikt voor het wachtwoord, nu altijd een x. Wachtwoorden staan nu in het `shadow` bestand.
3. nummerieke ID van de gebruiker
4. nummerieke ID van de primaire groep van de gebruiker
5. extra informatie over de gebruiker, met komma's gescheiden. Heet ook wel het GECOS-field en kan dan de volgende informatie bevatten:
   (a) Volledige naam van de gebruiker
   (b) Adres gegevens van de gebruiker (gebouw en kamernummer)
   (c) Werk telefoonnummer
   (d) Thuis telefoonnummer
   (e) Overige contact informatie (fax, prive e-mail adres, pager, social media)
6. de home-directory van de gebruiker
7. de shell die wordt opgestart als de gebruiker inlogt

Elke gebruiker is ook lid van minimaal 1 groep, de primaire groep zoals opgegeven in `/etct/passwd`. Op sommige systemen is dat de groep **users** op andere systemen is dat een andere groep. Om te zien van welke groepen je lid bent kan je `id` gebruiken.

```
$ id
```

De output geeft weer dat je maar één UID hebt en één of meer GID's. Je kan dus lid zijn van meer groepen. De eerste groep is de standaard (default) groep waarvan je lid bent.

In de output zie je ook dat elk UID en elke GID eigenlijk een nummer is. Computers kunnen alleen met getallen werken, terwijl wij mensen beter

met namen om kunnen gaan. Vandaar dat het besturingssysteem steeds een vertaling maakt van naam naar getal.

Zoekt uit hoe je `id` alleen het nummerieke ID terug kunt laten geven en hoe alleen de naam van de primaire groep waarvan je lid bent.

De 'database' met groep informatie vind je in `/etc/group`. Je kan de naam van bijvoorbeeld je primaire groep terug vinden door een `grep` te doen op :GID:, dus in het voorgaande voorbeeld zou dat betekenen:

```
$ grep :1000: /etc/group
```

de dubbele punten om het id is om te voorkomen dat je bijvoorbeeld ook groep ID 10000 of 10001 terug krijgt. User Personal Group of UPG is een gecombineerd gebruik van de user-ID en de group-ID voor het opslaan van data. Er is ook een groepsnaam met de gebruikersnaam.

In de voorgaande voorbeelden zag het gebruik van UPG. De gebruiker dennis heeft een UID van 1000 een GID van 1000 en beide zijn gekoppeld aan de naam dennis. Dus de groepsnaam is gelijk aan de gebruikers naam. De gebruiker is dus niet lid van de groep **users** waar op andere systemen elke gebruiker lid van is.

Door gebruik te maken van een groep **users** waar elke gebruiker lid van is maak je het delen van informatie met andere gebruikers een stuk makkelijker omdat iedereen een gedeelde groep heeft. Data die toegankelijk is voor de groep **users** is dus toegankelijk voor iedereen.

UPG is meer gericht op veiligheid. Omdat elke gebruiker alleen in zijn eigen groep zit kan data alleen gedeeld worden als er een groep aangemaakt wordt waarin bepaalde gebruikers worden toegevoegd. Die groep kan dan data met elkaar delen. Verder is alle data die je maakt alleen toegankelijk voor jezelf.

9.2 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.

9.2.1 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
```

9.2.2 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
```

9.3 Gebruikersbeheer

Om gebruikers toe te voegen aan het systeem zijn er twee tools. De eerste is `useradd` en de andere is `adduser`. Het programma `useradd` is een low-level tool, de standaard manier om gebruikers aan te maken is via `adduser` en dat is dan ook wat we gaan gebruiken.

```
$ sudo adduser eengebruiker
```

Om gebruikers te kunnen aanmaken hebben we root-rechten nodig, dus we gebruiken het `sudo` commando.

Het `adduser` commando maakt bijna alles automatisch aan. Het enige dat je hoeft te doen is te vertellen wat het wachtwoord van de gebruiker is en welke informatie er in het GECOS-veld terecht moet komen.

Je mag aan `adduser` ook meegeven dat zaken anders moeten zijn. Bijvoorbeeld dat een gebruiker een andere shell gebruikt dan bash. Dat kan door de optie `-shell /bin/chsh` mee te geven. Lees de man-page van `adduser` eens door met wat er nog meer mogelijk is.

Als op het systeem UPG gebruikt wordt dan zal je zien dat er ook gelijk een groep voor de gebruiker aangemaakt is. Gebruik `id` om te zien wat de ID is van de nieuwe gebruiker.

We gaan nu een extra groep aanmaken:

```
$ sudo addgroup specialgroup
```

deze groep is nu toegevoegd aan het `/etc/group` bestand.

Om een gebruiker toe te voegen aan een groep, gebruiken we `gpasswd`. `gpasswd` is een tool om groepen te beheren.

Om een gebruiker aan een groep toe te voegen gebruiken we:

```
$ sudo gpasswd –a eengebruiker specialgroup
```

om diezelfde gebruiker weer uit de groep te halen gebruiken we:

```
$ sudo gpasswd –d eengebruiker specialgroup
```

Om de instellingen van een gebruiker te wijzigen gebruiken we `usermod`.

Er zijn vele zaken die we kunnen instellen voor een gebruiker. De man-page van `usermod` vermeldt ze allemaal.

Om een groep weer te verwijderen is er `delgroup`. Het verwijderen van een groep betekent dat gebruikers uit de groep verdwijnen, maar eventuele bestanden op disk die van deze groep zijn veranderen niet. Je houdt dus bestanden over die nummeriek nog van de groep zijn, maar waar geen groepsnaam voor is. Dus alleen de owner kan er nog bij (en root natuurlijk). Het verwijderen van gebruikers kan heel simpel gedaan worden met `deluser`. Het heeft echter wel wat gevolgen. Als een gebruiker niet meer bestaat op het systeem dan kunnen er nog wel bestanden zijn die van deze gebruiker waren (zoals in zijn of haar home-directory). Nummeriek zijn deze bestanden dan ook nog van het UID van de voormalige gebruiker. Alleen root kan nog bij deze bestanden, maar wat ook kan gebeuren is dat het UID opnieuw toegewezen wordt aan een nieuwe gebruiker die aangemaakt wordt. Deze gebruiker wordt dan opeens de eigenaar van de nog aanwezige bestanden. Zorg er dus voor dat de bestanden van eigenaar veranderd zijn voordat je een gebruiker verwijderd.

Hoofdstuk 10

Toegangsrechten op bestanden en directories

Elk besturingssysteem dat meer dan één gebruiker kent heeft een manier nodig om ervoor te zorgen dat de twee gebruikers niet bij elkaars bestanden kunnen als ze dat niet willen. Ook moeten gebruikers niet bij de bestanden van de beheerder (root) kunnen komen. Er moet dus door het systeem bijgehouden worden wie welke rechten heeft op een bestand.

Bij het ontwerp van Unix was de centrale gedachte dat alles binnen het systeem gerepresenteerd werd door het idee van een bestand: 'Everything is a file'. Dus niet alleen documenten zijn bestanden, maar ook aangesloten printers, harddisks, etc. Omdat alles een bestand is kan je met rechten op bestanden ook bepalen wie toegang heeft tot bepaalde stukken hardware.

Dit hoofdstuk gaat over de rechten op Unix-achtige systemen zoals Linux. Met `ls -l` kan je een lijst van bestanden opvragen die meer weergeeft dan alleen de bestandsnaam. De output van -l -l zou er zo uit kunnen zien:

```
total 31657256
drwxr-xr-x  3 dennis dennis       4096 Jan  9 08:38  Apps
-rw-r--r--  1 dennis dennis 30752636928 Mar  5  2020
    bootdisk_32G_20200305.img
drwxr-xr-x  2 dennis dennis       4096 Jan  8  2020  Desktop
drwxr-xr-x 10 dennis dennis       4096 Jul 13  2022  Documents
drwxr-xr-x  4 dennis dennis      16384 Jan 19 13:21  Downloads
-rw-r--r--  1 dennis dennis         13 Jan 22  2020  hello.txt
-rw-r--r--  1 dennis dennis        131 Jan 22  2020  HELLO.txt
drwx------  2 root   root        16384 Sep 16  2019  lost+found
drwxr-xr-x  5 dennis dennis       4096 Mar 15  2022  Nextcloud
-rw-r--r--  1 dennis dennis     137665 Mar  3  2022  output.pdf
drwxr-xr-x  4 dennis dennis       4096 May 26  2020  PGP
drwxr-xr-x  3 dennis dennis       4096 Apr  5  2022  Pictures
drwxr-xr-x  4 dennis dennis       4096 Sep 30  2019  Projects
```

```
drwxr-xr-x  3 dennis dennis      4096 Feb  1  2022  src
-rw-r--r--  1 dennis dennis       420 Mar 31  2020  Teams.txt
drwxr-xr-x  2 dennis dennis      4096 Sep 25  2019  Templates
-rw-r--r--  1 dennis dennis        65 Apr  6  2022  test.py
-rwxr--r--  1 dennis dennis       345 Apr 16  2020  test.sh
drwxr-xr-x  2 dennis dennis      4096 Sep 23 15:32  tmp
drwx------ 24 dennis dennis      4096 Jan 18 13:23 'VirtualBox VMs'
```

De kolommen van links naar rechts geven weer:

- De rechten op een bestand,
- Het aantal links naar dit bestand
- De eigenaar van het bestand
- De groepseigenaar van het bestand
- De grootte van het bestand
- De maand van de laatste wijziging
- De dag van de laatste wijziging
- Het jaar van de laatste wijziging (of het tijdstip van de laatste wijziging als het minder dan een jaar geleden is)
- De naam van het bestand

10.1 Bestandstypen

10.2 src/bestandstypen

10.3 De eigenaar

Met `chown` kan de eigenaarschap van een bestand wijzigen. Om dit te kunnen testen moeten we een aantal zaken regelen:

- Maak een groep aan met de naam **samen**
- Voeg jezelf en de gebruiker eengebruiker toe aan deze groep
- Maak een directory aan `/home/samen`

Nu gaan we ervoor zorgen dat de groep samen toegang heeft tot de directory samen en dat jezelf de hoofd eigenaar wordt. We beginnen met het laatste:

```
$ sudo chown $(id –un) /home/samen
```

We hebben in dit commando een extraatje toegevoegd. We kunnen een variabele gebruiken om de gebruikers naam in te zetten en die gebruiken bij `chown`. Er bestaat echter ook een mogelijkheid om in de shell direct een commando aan te roepen en deze als variabele te gebruiken en dat is wat we hier gedaan hebben. We hebben `id -un` aangeroepen (wat onze gebruikersnaam terug geeft) en deze hebben we gebruikt als optie aan `chown`. Dus eigenlijk staat er `sudo chown username /home/samen`. Bekijk met `ls -l` het resultaat.

Nu gaan we zorgen dat de beide gebruikers gebruik kunnen maken van deze directory. Eerst moeten we zorgen dat de groep **samen** de groepseigenaar wordt van de directory:

```
$ sudo chown .samen /home/samen
```

Door een punt voor **samen** te zetten gevevn we aan dat we de groepseigenaarschap willen wijzigen. Bekijk met `ls -l` het resultaat.

Beide commando's hadden we ook in één keer kunnen doen:

```
$ sudo chown dennis.samen /home/samen
```

10.4 read, write and execute

De rechten op een bestand zijn opgedeeld in drie blokken. Elk blok kan de waarden r, w, en x bevatten. Er zijn dus in totaal 9 posities (rwxrwxrwx). De mogelijke rechten zijn:

r Read

w Write

x Execute

Elk blokje heeft zijn eigen betekenis:

Type	Owner	Group	Other
d	rwx	rwx	rwx

Er zijn dus rechten te vergeven voor de eigenaar, voor de groep en voor de rest van de wereld.

Voor normale bestanden geldt dat je met read rechten een bestand mag openen en dus het kunt lezen, met write rechten mag je bestand ook schrijven

en dus wijzigen en met execute rechten mag je een bestand opstarten, dat is natuurlijk alleen handig als je een bestand ook daadwerkelijk op mag starten zoals een script of programma.

Voor directories zijn de regels even anders. Met leesrechten mag je zien welke bestanden er in een directory staan en mag je deze lezen, je kan dus `ls` gebruiken en een bestand openen in de directory, met schrijfrechten mag je nieuwe bestanden aanmaken en met execute het je daadwerkelijk toegang tot de directory kortom je kunt `cd` gebruiken om in de directory te komen.

Computers zijn slecht in namen, maar goed in nummers, dus de rechten r, w en x moeten omgezet worden naar een getal waarmee de computer kan werken. Omdat computers heel goed zijn in binair zijn de rechten omgezet naar binaire getallen. Een 1 betekent dat het recht gegeven is, een 0 zegt dat het recht niet gegeven is. 101 betekent dus leesrechten en execute-rechten. Deze binaire manier van tellen kan ook decimaal geschreven worden 101 is dan 5. Een blok van rechten voor eigenaar, groep en de wereld zou er zo uit kunnen zien: rwxr-x-r–. Omgerekend naar binair is dat 111 101 100 en dat per stukje omgezet naar decimaal is 754.

Om de rechten op een bestand te wijzigen is er het commando `chmod`. Je kan `chmod` gebruiken om de rechten op bestanden te wijzigen door gebruik te maken van read, write en execute of door gebruik te maken van de decimale waarden van de rechten. Een voorbeeld van het gebruik van de decimale waarden zou voor de directory **samen** er zo uit kunnen zien:

```
$ sudo chmod 777 /home/samen
```

We hebben nu de eigenaar, de groep en de wereld alle rechten gegeven, dus de beide gebruikers in de groep **samen** kunnen nu bij alle documenten die ze in deze directory aanmaken.

Helaas hebben we ook alle rechten gegeven aan Other. Dat betekent dat de hele wereld bij alle documenten kan. We kunnen met chmod ook rechten afnemen. Gebruik eens:

```
$ sudo chmod o–x /home/samen
```

Na een `ls -l` zal je zien dat van other (o) de execute-rechten (x) verdwenen zijn. Dat kunnen we ook met meerdere rechten doen:

```
$ sudo chmod g–w,o–rw /home/samen
```

We ontnemen hier van other de read en write rechten en van de group rechten verwijderen we de schrijfrechten. Het plus-teken kunnen we gebruiken om rechten toe te kennen:

```
$ sudo chmod g+w /home/samen
```

zorgt ervoor dat de groep weer schrijfrechten heeft.

10.5 Het 4de-bit

De rechten r, w en x zijn elke keer blokjes van 3-bits. Het totaal is dus 3x3 is 9 bits lang. Dat is een raar getal in de computerwereld en dan ook niet helemaal correct, eigenlijk is het blok 12-bits lang en bestaat het uit 4x3 bits. De triplet 777 is dus eigenlijk een quadlet 7777. De eerste 7 wordt gebruikt om extra zaken in te coderen. De rechten op een bestand zonder extra functionaliteit is dus eigenlijke 0777, maar daar laten we de 0 meestal weg.

Met de extra rechten kunnen we de volgend functionaliteit weergeven:

- SUID bit
- SGID bit
- Sticky bit

10.5.1 SUID-bit

Het SUID-bit is het meest linkse bit uit de reeks en heeft dus een decimale waarde van 4. Het is de bit die hoort bij de user dus kan je het SUID-bit ook setten door `u+s` te gebruiken. Voorbeelden:

```
$ mkdir SUID.d
$ touch SUID.txt
$ chmod u+s SUID.d
$ chmod 4777 SUID.txt
$ ls -ld SUID*
drwsr-xr-x 1 dennis dennis  4096 Feb  7  2022 SUID.d
-rwsrwxrwx 1 dennis dennis 63960 Feb  7  2022 SUID.txt
```

Het eerste rwx blokje is nu veranderd in rws om aan te geven dat het SUID-bit gezet is.

Het bit op een bestand zorgt ervoor dat, als je het bestand kunt opstarten, de applicatie draait onder de username van de eigenaar. Dus als een bestand als eigenaar heeft root.admin en het SUID-bit is gezet dan zal bij opstarten het programma draaien met root-rechten. Het voordeel is dat gewone gebruikers zo programma's kunnen opstarten met rechten die ze normaal niet hebben. Het nadeel is dat er een security-lek zou kunnen ontstaan, dus je moet heel voorzichtig zijn met deze rechten.

Een voorbeeld op een Linux systeem waar het SUID-bit gebruikt wordt is op het `passwd` programma. Een gebruiker moet instaat zijn om zijn wachtwoord te wijzigen, terwijl het `/etc/shadow` bestand alleen lees en schrijfbaar is door root.

Het bit op een directory heeft geen betekenis in GNU/Linux.

10.5.2 SGID-bit

Het SGID-bit is het tweede bit uit de reeks en is dus decimaal: 2. Het behoort bij de groepsrechten en kan dus gezet worden met `g+s`. Voorbeelden

```
$ mkdir SGID.d
$ touch SGID.txt
$ chmod g+s SGID.d
$ chmod 2777 SGID.txt
$ ls -ld SGID*
drwxr-sr-x 2 dennis dennis 4096 Jan 24 09:28 SGID.d
-rwxrwsrwx 1 dennis dennis    0 Jan 24 09:28 SGID.txt
```

Het SGID-bit op een bestand zet de effectieve groep waaronder een applicatie draait. Dus net als wat de SUID-bit doet voor de gebruiker doet het SGID-bit voor de groep,

Het SGID-bit op directories betekent dat de groep eigenaarschap wordt doorgegeven aan nieuwe bestanden of directories die aangemaakt worden in de directory. Dus als we een directory hebben met het SGID-bit en de groepseigenaar van de directory is **samen** en we maken in die directory een nieuwe bestand aan dan wordt de groepseigenaar weer **samen** ongeacht de groep waarin de gebruiker zit die het bestand aanmaakt. Natuurlijk moet een van de groepen van die gebruiker wel **samen** zijn.

10.5.3 Sticky-bit

Het Sticky-bit kan gezet worden met de decimale waarde 1 of met `o+t`. Voorbeelden:

```
$ mkdir sticky.d
$ touch sticky.txt
$ chmod o+t sticky.d
$ chmod 1777 sticky.txt
$ ls -dl sticky*
drwxr-xr-t 2 dennis dennis 4096 Jan 24 10:04 sticky.d
-rwxrwxrwt 1 dennis dennis    0 Jan 24 10:04 sticky.txt
```

Bij het rechtenblok van other is de x vervangen door een t.

Voor bestanden heeft het Sticky-bit in Linux geen betekenis.

Als het Sticky-bit gezet is op een directory dan betekent dat alleen de eigenaar, de eigenaar van de directory en root het bestand kunnen weggooien en hernoemen. Zelfs als iemand in de juiste groep zit en de groep heeft schrijfrechten dan nog heeft die persoon niet de rechten. Een voorbeeld van het gebruik van het Sticky-bit is het `/tmp` directory.

Hoofdstuk 11

Systeem inventarisatie

Niet elk bedrijf heeft een CMDB (Configuration Management Database) of soms ontbreken er nog zaken in de database. Servers staan vaak in een serverruimte en zijn niet makkelijk physiek toegankelijk en ze mogen zeker niet uit om te zien wat er aan hardware in het systeem zit. In dit hoofdstuk ga je leren hoe je de hardware informatie van een systeem verzamelt met de commando's op de Linux command prompt.

11.1 System versienummers

Een Linux systeem is opgebouwd uit een kernel, extra software van bijvoorbeeld het GNU-project en nog veel meer software van verschillende plekken. En toch levert een distributie bouwer een systeem af met één enkel versienummer. Voor de verschillende software pakketten zijn er een aantal standarden in omloop om het versienummer te achterhalen. Bij de meeste commando's kan je `-v` of `-version` gebruiken als optie het versienummer te achterhalen. Bij veel commando's werken beide opties, bij sommige alleen één van de twee. De lange versie met `-version` is een GNU-optie, op het oorsponkelijke UNIX systeem bestonden geen lange opties. Dus de meeste GNU-commando's ondersteunen deze lange optie. Een heel enkele keer wordt `-v` voor iets anders gebruikt, dus het is verstandig om eerst de manual-pagina van het commando te raadplegen om te zien of de optie inderdaad het versienummer weergeeft.

In de rest van deze paragraaf gaan we het hebben over hoe je het versienummer van de distributie en Linux achterhaalt.

11.1.1 Distributie versie

Om op een draaiend systeem te achterhalen welke distributie er geïnstalleerd is valt vaak niet mee. Het begon ermee dat elke distributie zijn informatie op zijn eigen plek bewaarde. Zo hadden de verschillende systemen hun eigen bestanden in de `/etc/` directory:

SUSE /etc/SUSE-brand

Debian /etc/debian_version

RedHat /etc/redhat-release

en ook de inhoud van deze bestanden was niet consistent. Er is een poging gedaan om dit te verbeteren. Er is nu één bestand dat op elk systeem aanwezig is en dat is `/etc/os-release`. Een vergelijking tussen de inhoud op een Debian systeem en op een SUSE-systeem laat al gelijk zien dat we er nog niet helemaal zijn:

```
PRETTY_NAME="Debian GNU/Linux 10 (buster)"
NAME="Debian GNU/Linux"
VERSION="10 (buster)"
VERSION_ID="10"
VERSION_CODENAME=buster
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
```

```
PRETTY_NAME="openSUSE Leap 15.1"
NAME="openSUSE Leap"
VERSION="15.1"
ID="opensuse-leap"
ID_LIKE="suse opensuse"
HOME_URL="https://www.opensuse.org"
BUG_REPORT_URL="https://bugs.opensuse.org"
CPE_NAME="cpe:/o:opensuse:leap:15.1"
```

Het belangrijkste is dat we nu alleen `/etc/os-release` nodig hebben om te achterhalen op welke distributie we aan het werk zijn.

11.1.2 Kernel versie

Informatie over de kernel die actief is kan verkregen worden met het `uname` commando. `uname` heeft verschillende opties (zie de manual-page). De belangrijkste optie voor ons is de `-r` optie, want die verteld het release nummer van de kernel:

```
$ uname -r
4.19.0-13-amd64
```

in de release is het deel voor het eerste streepje (-) het versie nummer van de kernel-broncode die gebruikt is. De rest is afkomstig van de bouwers van de distributie en die informatie kan verschillen per distributie bouwer.

Met de `-v` optie kan je ook de datum achterhalen waarop deze kernel gecompileerd is:

```
$ uname -v
#1 SMP Debian 4.19.160-2 (2020-11-28)
```

in dit geval dus 28 november 2020.

11.2 Het moederbord

11.2.1 CPU

Om te achterhalen welke processor er in een machine zit is er het commando `lscpu`. Het geeft snel veel informatie over de processor:

```
$ lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
Address sizes:       39 bits physical, 48 bits virtual
CPU(s):              8
On-line CPU(s) list: 0-7
Thread(s) per core:  2
Core(s) per socket:  4
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               60
Model name:          Intel(R) Core(TM) i7-4810MQ CPU @ 2.80GHz
Stepping:            3
CPU MHz:             1291.340
CPU max MHz:         3800.0000
CPU min MHz:         800.0000
BogoMIPS:            5586.67
Virtualization:      VT-x
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            6144K
NUMA node0 CPU(s):   0-7
```

```
Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
    pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm
    constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid
    aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma
    cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes
    xsave avx f16c rdrand lahf_lm abm cpuid_fault epb invpcid_single pti ssbd ibrs
    ibpb stibp tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1
    avx2 smep bmi2 erms invpcid xsaveopt dtherm ida arat pln pts flush_l1d
```

Informatie per aanwezige CPU kan verkregen worden via `/proc/cpuinfo` maar dit levert over het algemeen niet meer informatie op dan wat `lscpu` al gegeven heeft. Met `cat` kan je de inhoud van `/proc/cpuinfo` weergeven.

11.2.2 RAM

Het geheugengebruik op een machine is belangrijk om te weten. Er zijn dan ook verschillende commando's die hier informatie over geven. In deze paragraaf zullen we het hebben over de commando's die je vertellen hoeveel geheugen er in de computer zit. Het belangrijkste is doe de Linux-kernel tegen het geheugen aankijkt en dat kunnen we zie door gebruikt te maken van `/proc/meminfo` bestand. Met `cat` of `less` kunnen we dat bestand lezen en dan zien we heel veel informatie over het geheugen. De belangrijkst informatie voor ons op dit moment is de totale hoeveelheid geheugen (RAM) die er in de machine zit en hoeveel er in gebruik is. Een compactere manier om dit weer te geven is door gebruik te maken van het `free` commando:

```
$ free
              total        used        free      shared  buff/cache   available
Mem:       32538364    18631520      591412     2071772    13315432    11378676
Swap:      15236492       63816    15172676
```

`free` heeft nog een andere handige functie en dat is dat het deze informatie met een bepaalde interval, bijvoorbeeld elke 10 seconden, kan weergeven. Zo kan je een drukke server in de gaten houden. Om dit te doen gebruik je `free -s 10`.

11.2.3 BIOS

De BIOS of EUFI is verantwoordelijk voor het opstarten van de computer, het moet dus heel veel weten van de hardware in een systeem. Als we de BIOS zouden kunnen uitvragen over ons systeem dan zou dat heel veel informatie kunnen opleveren, potentieel zelfs heel gevoelige informatie. Je kan dan dit dan ook niet als gewone gebruiker. Je moet root zijn om `dmidecode` te kunnen gebruiken. In de gegeven voorbeelden zullen we dna ook `sudo` gebruiken.

De totale informatie die aanwezig is in de DMI-tabel is heel veel, dus het is handiger om er alleen de stukjes uit te halen die we nodig hebben. Dit doen we door aan `demidecode` de optie `-s` mee te geven met het onderdeel dat we zouden willen hebben. Als we bijvoorbeeld alleen de informatie over de BIOS versie willen hebben gebruiken we:

```
$ sudo dmidecode -s bios-version
GNET88WW (2.36 )
```

Als je alleen de `-s` optie meegeeft en verder niets dan zie je welke keywords er bestaan voor `-s`.

Als we iets meer willen weten van het BIOS kunnen we ook meer informatie op vragen. Alle informatie over het systeem is opgedeeld in types. Met `-t` kan je alle informatie van een type opvragen, zonder een keyword krijg je een lijst te zien van beschikbare types. Alle BIOS informatie valt onder het keyword bios.

```
$ sudo dmidecode -t bios
# dmidecode 3.2
Getting SMBIOS data from sysfs.
SMBIOS 2.7 present.

Handle 0x002A, DMI type 13, 22 bytes
BIOS Language Information
        Language Description Format: Abbreviated
        Installable Languages: 1
                en-US
        Currently Installed Language: en-US

Handle 0x0040, DMI type 0, 24 bytes
BIOS Information
        Vendor: LENOVO
        Version: GNET88WW (2.36 )
        Release Date: 05/30/2018
        Address: 0xE0000
        Runtime Size: 128 kB
        ROM Size: 12288 kB
        Characteristics:
                PCI is supported
                PNP is supported
                BIOS is upgradeable
                BIOS shadowing is allowed
                Boot from CD is supported
                Selectable boot is supported
                ACPI is supported
                USB legacy is supported
                BIOS boot specification is supported
                Targeted content distribution is supported
```

```
        UEFI is supported
    BIOS Revision: 2.36
    Firmware Revision: 1.14
```

Je ziet dat we zo al heel veel meer informatie naar boven halen.

`dmidecode` commando waarmee je veel informatie kan vinden. Lees de manual-pagina eens door en speel eens met de `-t` en `-s` opties om ermee vertrouwd te raken.

11.3 Extensie bussen

11.3.1 PCI

De PCI en PCIe bussen kunnen ook vertellen wat er allemaal aan zit. `lspci` vertelt ons wat er allemaal aan de PCI-bus hangt.

```
$ lspci
00:00.0 Host bridge: Intel Corporation Xeon E3−1200 v3/4th Gen Core Processor
    DRAM Controller (rev 06)
00:01.0 PCI bridge: Intel Corporation Xeon E3−1200 v3/4th Gen Core Processor PCI
    Express x16 Controller (rev 06)
00:02.0 VGA compatible controller: Intel Corporation 4th Gen Core Processor
    Integrated Graphics Controller (rev 06)
00:03.0 Audio device: Intel Corporation Xeon E3−1200 v3/4th Gen Core Processor HD
    Audio Controller (rev 06)
00:14.0 USB controller: Intel Corporation 8 Series/C220 Series Chipset Family USB
    xHCI (rev 04)
00:16.0 Communication controller: Intel Corporation 8 Series/C220 Series Chipset
    Family MEI Controller #1 (rev 04)
00:19.0 Ethernet controller: Intel Corporation Ethernet Connection I217−LM (rev 04)
00:1b.0 Audio device: Intel Corporation 8 Series/C220 Series Chipset High Definition
    Audio Controller (rev 04)
00:1c.0 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express
    Root Port #1 (rev d4)
00:1c.1 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express
    Root Port #2 (rev d4)
00:1c.2 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express
    Root Port #3 (rev d4)
00:1c.4 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express
    Root Port #5 (rev d4)
00:1d.0 USB controller: Intel Corporation 8 Series/C220 Series Chipset Family USB
    EHCI #1 (rev 04)
00:1f.0 ISA bridge: Intel Corporation QM87 Express LPC Controller (rev 04)
00:1f.2 SATA controller: Intel Corporation 8 Series/C220 Series Chipset Family 6−port
    SATA Controller 1 [AHCI mode] (rev 04)
00:1f.3 SMBus: Intel Corporation 8 Series/C220 Series Chipset Family SMBus
    Controller (rev 04)
```

```
01:00.0 VGA compatible controller: NVIDIA Corporation GK107GLM [Quadro
    K1100M] (rev a1)
01:00.1 Audio device: NVIDIA Corporation GK107 HDMI Audio Controller (rev a1)
02:00.0 SD Host controller: O2 Micro, Inc. SD/MMC Card Reader Controller (rev 01)
03:00.0 Network controller: Realtek Semiconductor Co., Ltd. RTL8192EE PCIe
    Wireless Network Adapter
```

Handige informatie die we hieruit kunnen halen is de grafische controller, in dit geval één van NVIDIA, een audio controller en de netwerkkaart.

We zien ook dat de chipset USB ondersteunt.

11.3.2 USB

Met `lsusb` kunnen we informatie opvragen over de USB-bus.

```
$ lsusb
Bus 001 Device 002: ID 8087:8000 Intel Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 002: ID 0781:5583 SanDisk Corp. Ultra Fit
Bus 003 Device 056: ID 125f:dd1a A−DATA Technology Co., Ltd.
Bus 003 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 002 Device 002: ID 69a7:9803
Bus 002 Device 004: ID 04f2:b39a Chicony Electronics Co., Ltd
Bus 002 Device 003: ID 0bda:8761 Realtek Semiconductor Corp.
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

We zien hier ook veel overbodig informatie. Op Bus 3 device 2 en op Device 56 zit een USB-memory stick, maar dat moet je al kunnen herkennen aan de leveranciersnaam om dat eruit te halen. In de paragraaf over disken zullen we hier een betere methode voor laten zien. `lsusb` is wel handig als er bijvoorbeeld een digitale camera of printer aan het systeem hangt. Dan levert het vaak nuttige extra informatie op.

11.3.3 SCSI

Harddisks, USB-sticks en andere opslagsystemen hangen in Linux onder het SCSI-subsystem, ongeacht of het een SCSI-apparaat is. Dus ook SATA-harddisken en SSDs zijn onder het SCSI subsystem terug te vinden.

```
$ lsscsi
[0:0:0:0] disk ATA     Samsung SSD 860  1B6Q  /dev/sda
[6:0:0:0] disk SanDisk Ultra Fit        1.00  /dev/sdb
[7:0:0:0] disk ADATA   USB Flash Drive  1100  /dev/sdc
```

Hier zien we al duidelijker dat in dit systeem één SSD zit en twee USB-sticks.

11.4 Harddisks

Harddisks, SSDs, USB-sticks heten onder Linux block-devices. Het zijn opslag apparaten die hun data schrijven in blocks (sectoren). Vroeger hadden we IDE-harddisks en dat waren de eerste harddisks die ondersteunt werden door Linux. Deze harddisks kregen de device naam hd van harddisk gevolgt door een letter (a t/m z), dus hda was de eerste harddisk, hdb de tweede, etc. Later kwam er ook support voor SCSI en deze disken werden sd genoemd met een letter, daarmee werd sda de eerste SCSI harddisk en sdb de tweede.

In de moderne Linux-kernels worden bijna alle block-devices aangestuurd door het SCSI-systeem. Dus alle disken beginnen met sd, ongeacht of het een echte SCSI-disk is.

Met `lsblk` krijg je een overzicht van de block-devices in je systeem.

```
$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 931.5G  0 disk
|−sda1   8:1    0 931.5G  0 part /home/dennis
sdb      8:16   1  28.7G  0 disk
|−sdb1   8:17   1  14.1G  0 part /
|−sdb2   8:18   1     1K  0 part
|−sdb5   8:21   1  14.5G  0 part [SWAP]
sdc      8:32   1  28.9G  0 disk
|−sdc1   8:33   1  28.9G  0 part
```

We zien dat we naast de sda, sdb en sdc disk ook nog disken hebben met een nummer er achter. Dat zijn de partities van de disk. De disk sdb heeft dus 3 partities.

Om per partitie te kunnen zien wel bestandssysteem er gebruikt wordt gebruiken we de `-fs` optie. Omdat veel extra informatie oplevert hebben we voor dit boek een deel weggelaten zodat de tabel op een pagina past. Probeer het commando ook uit op je eigen systeem zodat je alle informatie ziet.

```
$ lsblk −fs
NAME  FSTYPE  LABEL
sda1  ext4
|−sda
sdb1  ext4    Debian 6.0.2.1 M−A 1
|−sdb iso9660 Debian 6.0.2.1 M−A 1
sdb2  iso9660 Debian 6.0.2.1 M−A 1
|−sdb iso9660 Debian 6.0.2.1 M−A 1
sdb5  swap    Debian 6.0.2.1 M−A 1
|−sdb iso9660 Debian 6.0.2.1 M−A 1
sdc1  vfat    ADATA UFD
|−sdc
```

We zien dat de partitie sda1 een ext4 bestandsysteem heeft en dat de partitie sdc1 vfat gebruikt.

Om te zien welke bestandssystemen je systeem zou kunnen ondersteunen kan je het volgende doen:

```
$ ls /usr/lib/modules/‘uname –r‘/kernel/fs
```

dit vraagt uit de directory met de release van de huidige kernel de lijst met bestandssysteem modules op. Linux kent verscheidene tools om een harddisk te beheren, één van de oudste en meest gebruikte is `fdisk`. `fdisk` heeft geen grafische interface, maar wel een character gebaseerde interface. Naast `fdisk` zijn er ook `cfdisk` en `sfdisk`. De `cfdisk` tool heeft een iets vriendelijkere interface, maar er kan iets minder mee en `sfdisk` is vooral in scripts.

Met het commando

```
$ sudo fdisk –l
```

kan je een overzicht krijgen van alle disks en partities in het systeem inclusief hun grote. Vaak willen we ook weten wat het gebruik is van harddisk, moeten we al een grotere kopen of kunnen we nog even voort, hiervoor hebben we het disk free commando: `df`. Een handige optie bij `df` is de `-h` optie die de waarden terug geeft in een human readable format, ofwel een door mensen leesbaar formaat. Je krijgt de waarden dan terug in mega- en gigabytes. Er zijn van die momenten dat je naast dat je wilt weten hoeveel diskruimte er vrij is ook wilt weten wie nu eigenlijk de grootste gebruiker is. Om dit te bepalen moeten we het disk gebruik, ofwel, disk usage, bepalen met behulp van het `du` commando. Als je `du` opstart zonder opties geeft het een opsomming van de grootte van de bestanden in de directory waarin je staat. Daar zou je natuurlijk ook gewoon `ls -l` voor kunnen gebruiken, daar hebben we `du` niet voor nodig. `du` wordt handig met de optie `-s` die een optelling doet van alle bestanden in een directory. Een tweede handige optie is de `-h` optie die de optelling terug geeft in een human readable format. Met deze opties kunnen we bepalen hoe groot bijvoorbeeld een home-directory van een gebruiker is.

```
$ du –hs /home/dennis/
732G    /home/dennis/
```

Als je wilt zien wie de grootste gebruiker op je systeem is dan is het handig om een wildcard te gebruiken:

```
$ sudo du –hs /home/∗
732G    /home/dennis
16K     /home/hcftpuser
```

11.5 Netwerk

De netwerk-hardware zijn we bij `lspci` al tegen gekomen. De uitvoer van dat commando gaf 2 netwerkkaarten.

```
$ lspci | grep -i net
00:19.0 Ethernet controller: Intel Corporation Ethernet Connection I217-LM (rev 04)
03:00.0 Network controller: Realtek Semiconductor Co., Ltd. RTL8192EE PCIe
    Wireless Network Adapter
```

Er is dus een netwerkkaart en een draadloze netwerkadapter in ons systeem aanwezig.

Het `ip` commando gebruiken we om netwerk interfaces te configureren, maar we kunnen het ook gebruiken om informatie over een adapter op te vragen:

```
$ ip link show
```

Dit geeft de interface met hun netwerk-address (MAC).

11.6 Inventarisatie opdracht

Maak een document met daarin de systeem informatie van je computer (VM). De volgende informatie moet daarin opgenomen zijn:

Element	Waarde	Gebruikte commando met opties
CPU Modelnaam		
CPU Aantal		
RAM in Gibibytes		
BIOS release datum		
BIOS versie nummer		
Systeem manufacturer		
Systeem versie		
Serienummer moederbord		
Aantal partities op je opstartschijf		
Disk gebruik van je home-directory		
MAC adres van je netwerkkaart		

Hoofdstuk 12

Systeembeheer

Naast dat we als we nieuw zijn in een organisatie graag willen weten wat er draait, willen we als dagelijkse systeembeheerder natuurlijk ook weten hoe het er met het systeem voor staat. Vragen als is het systeem druk? Zijn er veel gebruikers? Zijn er processen met problemen? Dat zijn allemaal vragen waar we een antwoord op willen hebben.

Dit hoofdstuk gaat je helpen met het beantwoorden van deze en andere vragen die te maken hebben met het dagelijkse systeembeheer.

12.1 CPU gebruik

Je kan snel een indicatie krijgen van de drukte op een systeem door gebruik te maken van het commando `uptime`. `uptime` geeft op één reel een overzicht van de tijd, de tijd dat het systeem up is, het aantal gebruikers dat ingelogd is en de gemiddelde systeem belasting in de laatste 1, 5 en 15 minuten.

```
$ uptime
11:27:27 up 2 days, 5 min, 2 users, load average: 0.47,0.42,0.44
```

Als het getal bij het 1 minuut gemiddelde hoger is dan bij het 5 minuten gemiddelde, zoals in het bovenstaande voorbeeld, dan neemt de load op het systeem toe. Als het load gemiddelde bij 1 inuut lager is bij het 5 minuut gemiddelde dan neemt de belasting van het systeem af.

12.2 Geheugen gebruik

In het vorige hoofdstuk hebben we het gebruik van `free` gezien. Naast het totaal van het aanwezige geheugen geeft `free` ook het in gebruik zijnde geheugen aan:

```
$ free -g
              total        used        free      shared  buff/cache   available
Mem:             31          14           3           1          12          13
Swap:            14           0          14
```

Met de `-g` optie geeft `free` de waarden weer in gibi-bytes. In het bovenstaande voorbeeld hebben dus nog 3G vrij van de 31G die er in de machine zit.

12.3 Aanwezige gebruikers

Bij het gebruik van `uptime` hebben we gezien dat er twee gebruikers actief zijn op ons systeem. Wie zouden dat zijn? Juist! met het gebruik van `who` kunnen we dat achterhalen:

```
$ who
root     tty1         2021-02-09 18:07
dennis   tty7         2021-02-09 18:10 (:0)
```

We zien dat zowel root als dennis zijn ingelogd. Beide gebruikers zijn nog steeds ingelogd. De root gebruiker is 3 minuten eerder ingelogd dan de gebruiker dennis.

Met `last` zien we wie er ingelogd zijn en wie er ingelogd zijn geweest:

```
$ last
dennis tty7        :0           Tue Feb  9 18:10   still logged in
root   tty1                     Tue Feb  9 18:07   still logged in
dennis tty7        :0           Tue Feb  9 11:14 - 18:10  (06:56)
reboot system boot 4.19.0-13    Tue Feb  9 11:13   still running
```

De gebruiker dennis is op 9 februari om 11:14 ingelogd en om 18:10 weer uitgelogd.

12.4 Processen

Met `free` hebben we gezien wat het geheugen gebruik is en wat er nog vrij is. Het zou natuurlijk mooi zijn als we kunnen zien wat er geheugen gebruikt. Het geheugen wordt gebruikt door processen. Een proces is alles wat er in het geheugen zit en dat min of meer continue draait. Een overzicht van de aanwezige processen kan je krijgen met het `ps` commando. Het `ps` commando zonder opties laat alleen de processen zien van de gebruiker waaronder je bent ingelogd. Om alle processen te zien die op een systeem draaien is er de `-e`

optie. Dit is vaak een heel lange lijst met processen en die zijn gesorteerd op proces id:

```
$ ps -e | head -10
  PID TTY          TIME CMD
    1 ?        00:00:44 systemd
    2 ?        00:00:01 kthreadd
    3 ?        00:00:00 rcu_gp
    4 ?        00:00:00 rcu_par_gp
    6 ?        00:00:00 kworker/0:0H-kblockd
    8 ?        00:00:00 mm_percpu_wq
    9 ?        00:02:43 ksoftirqd/0
   10 ?        00:11:19 rcu_sched
   11 ?        00:00:00 rcu_bh
```

Met de `head -10` krijgen we de eerste 10 regels te zien. We zien dat proces nummer 1 (PID is Process ID, ofwel proces nummer) systemd is. Dat is op de meeste moderne systemen het geval. Nadat de kernel geladen is in het geheugen wordt systemd opgestart die dan zorg voor het opstarten van de rest van het systeem.

In de tweede kolom zien we dat bij TTY een vraagteken staat. Dat betekent dat het proces niet gekoppeld is aan een terminal en dus op de achtergrond rustig zijn werk aan het doen is. Gebruikers processen zijn gekoppeld aan een terminal:

```
PID   TTY          TIME CMD
18209 pts/10   00:00:00 vim
18313 pts/12   00:00:00 bash
18751 pts/11   00:00:03 vim
19027 pts/13   00:00:00 ps
```

Het laatste commando uit dit lijstje is onze eigen opdracht, `ps`, en de `vim` processen zijn de processen waarmee ik dit document aan het schrijven ben. Door aan `ps` de `-l` (long) optie mee te geven kunnen we ook zien wie de eigenaar van het proces is:

```
F S  UID   PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 18209  5870  0  80   0 -  4552 core_s pts/10   00:00:00 vim
0 S  1000 18313  1930  0  80   0 -  1748 -      pts/12   00:00:00 bash
0 S  1000 18751  8994  0  80   0 -  4533 core_s pts/11   00:00:05 vim
0 R  1000 19540 12035  0  80   0 -  2637 -      pts/13   00:00:00 ps
```

We zien dat we meerdere extra kolomn hebben gekregen waaronder de S van status. Voor ons `ps` proces staat daar een R, kortom dit proces is 'running'. De eigenaar van het proces is de gebruiker met UID 1000. En dat ben ik.

Wat opgevallen zal zijn is dat voor het `ps` commando we alleen ps zien en niet de opties. We hebben dus nog geen idee wat er echt gebeurd. Om dat te achterhalen moeten we de opties `-lef` gebruiken:

```
F S  UID     PID   PPID   C PRI  NI ADDR SZ WCHAN  TTY          TIME
     CMD
0 S dennis   18209  5870  0  80   0 -  4552 core_s 08:15 pts/10   00:00:00 vim
     src/Linux_deel2_CLI.tex
0 S dennis   18313  1930  0  80   0 -  1748 -      Mar16 pts/12   00:00:00 bash
0 S dennis   18751  8994  0  80   0 -  4567 core_s 08:35 pts/11   00:00:07 vim
     src/processen.tex
0 R dennis   20084 12035  0  80   0 -  2658 -      09:21 pts/13   00:00:00 ps -lef
```

Als eerste valt op dat lange regels doorgaan op de volgende regel, het UID 1000 is nu vertaald naar de naam van de gebruiker en van de commando's zien we nu ook met welke optie ze zijn opgestart. Bij het `ps` commando zien we dat de opties `-lef` worden weergegeven en bij de `vim` commando's zien we ook welke bestandsnamen er aan `vim` zijn meegegeven bij het starten. Er zijn momenten dat we een proces willen afsluiten omdat het klaar is met zijn taak of omdat we het niet meer willen gebruiken. Het commando om een draaiend proces te stoppen heet `kill`.

Het `kill` commando kan op twee manieren aan een proces vragen om te stoppen met werken. Het kan op de vriendelijke manier en op de botte manier. Op de vriendelijke manier krijgt een proces de mogelijkheid om alles wat hij nog aan het doen is af te ronden. Bij de botte manier is dat niet het geval. Het is verstandig om een proces te vragen op een vriendelijk manier af te sluiten zodat het proces netjes alles kan afhandelen. Op deze manier raakt er geen data verloren.

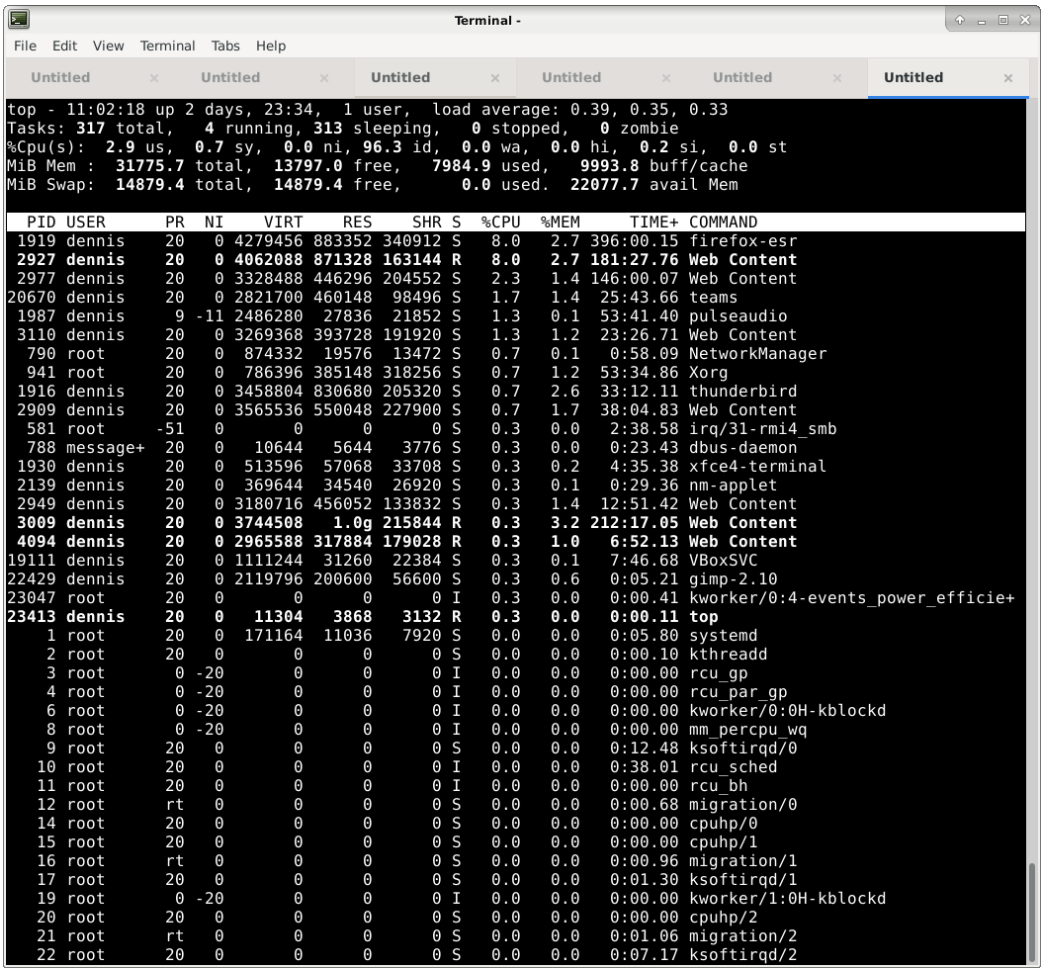
De nette manier is het vragen aan het proces om te stoppen, ofwel in Engels to terminate. Het signaal heet dan ook SIGTERM

```
$ kill -s SIGTERM firefox
```

De botte manier is het killen van het proces ofwel SIGKILL.

```
$ kill -s SIGKILL firefox
```

Als een systeem langzaam aan trager aan het worden is willen we graag weten wat er aan de hand is. Met een lijst zoals door `ps` gegeven kunnen we dat lastig monitoren, omdat we niet kunnen zien wat er in de tijd gebeurd. Met `top` kunnen we dat wel. `top` laat je in real-time zien wat er op je systeem gebeurt. Als je `top` opstart ziet het scherm er ongeveer uit zoals in figuur 12.1.



Figuur 12.1: Top

Met `q` verlaten we `top`.

`top` geeft je in één scherm heel veel informatie over je systeem en welke processen het meeste van je systeem vragen. Standaard sorteert `top` de processen op processor gebruik. We wie in het lijstje dat `firefox-esr` 8.0% CPU gebruikt en 2.7% van het geheugen. Met M en P (**let op** hoofdletters), kan je schakelen tussen een overzicht van het meeste processor (P) gebruik of het meeste memory (M) gebruik.

12.5 Logging

Het is belangrijk dat processen informatie kunnen doorgeven aan beheerders van systemen. Een proces moet kunnen melden dat er bijvoorbeeld iets fout

is gegaan. Met een grafische applicatie is dat geen probleem, die geeft dan een pop-up op het scherm. Een niet grafische applicatie kan een bericht op de commandline achterlaten, maar een proces dat in de achtergrond draait kan dat niet, die moet zijn meldingen ergens anders kwijt. Dit soort processen schrijven hun meldingen naar een log. Logging is de Engelse term voor vastleggen, opschrijven. Om die logging gestructureerd te doen, en niet elk proces zijn eigen keuze te laten maken, is er een directory genaamd `/var/log` waarin de log bestanden worden opgeslagen.

Elk proces mag zelf in deze directory schrijven of het mag gebruik maken een de logserver op een Linux systeem. Door gebruik te maken van de logserver worden logberichten op een gestandaardiseerde manier opgeslagen op het systeem.

12.5.1 syslog

Het proces dat op de achtergrond draait en dat verantwoordelijk is voor het schrijven van de verschillende logbestanden heet syslog, een afkorting van system logging. Het gebruik van syslog heeft als groot voordeel dat niet elke software ontwikkelaar alle code hoeft te schrijven om logbestanden te schrijven, hij hoeft alleen maar te weten hoe hij data moet aanlever bij de syslog server. Voor de systeembeheerder is het makkelijk omdat hij niet elk programma hoeft te vertellen waar deze zijn meldingen weg moet schrijven, die hoeft alleen maar de syslog-server te beheren en het laatste voordeel is dat de output van syslog altijd dezelfde is, dus de logbestanden worden nog makkelijker leesbaar ook.

Vele Linux distributies gebruiken rsyslog als hun syslog server. De belangrijkste logbestanden in de `/var/log` die beheerd worden door de syslog server zijn:

`messages` Alle logs

`mail` **(SuSE)** `maillog` **(RedHat) of** `mail.log` **(Debian)** Mail logging

`mail.info` Mail info messages

`mail.warn` Mail warning messages

`mail.err` Mail error messages

`daemon.log` **(Debian)** Berichten van Daemons

`auth.log` **(Debian) of** `secure` **(RedHat)** Authenticatie gerelateerde berichten

12.5.2 kernel berichten

De kernel wil ons soms ook wat berichten overbrengen. Zeker tijdens het opstarten van het systeem produceert de kernel vele berichten. De Linux kernel gebruikt een ringbuffer voor deze berichten. Een ringbuffer is een FIFO, First In, First Out, buffer, wat betekend dat berichten in de buffer worden opgeslagen tot de buffer vol is, nieuwere berichten drukken dan de oudste berichten uit de buffer. De standaard buffer grote is

`dmesg` is het commando om deze berichten uit te kunnen lezen. De bootberichten zijn zo belangrijk dat de meeste Linux distributes de ringbuffer na het opstarten leeglezen en deze opslaan als `/var/log/boot.log` of `/var/log/boot`. Want ook na het opstarten blijft de kernel naar deze ringbuffer schrijven en nieuwere berichten drukken dan uiteindelijk de berichten die gegenereerd zijn tijdens het opstarten uit de buffer.

Een belangrijke optie voor `dmesg` is de `-T` optie. De standaard timestamp in de `dmesg` output is niet lekker leesbaar, met `-T` krijgen een output die wel goed leest:

```
$ sudo dmesg -T
[Wed Mar 17 12:41:28 2021] CPU2: Package temperature/speed normal
[Wed Mar 17 12:41:28 2021] CPU1: Core temperature/speed normal
[Wed Mar 17 12:41:28 2021] CPU5: Package temperature/speed normal
[Wed Mar 17 12:41:28 2021] CPU4: Package temperature/speed normal
[Wed Mar 17 12:41:28 2021] CPU7: Package temperature/speed normal
[Wed Mar 17 12:41:28 2021] CPU6: Package temperature/speed normal
[Wed Mar 17 12:41:28 2021] CPU0: Core temperature/speed normal
[Wed Mar 17 12:41:28 2021] CPU3: Package temperature/speed normal
[Wed Mar 17 12:41:28 2021] CPU1: Package temperature/speed normal
[Wed Mar 17 12:41:28 2021] CPU0: Package temperature/speed normal
```

Hoofdstuk 13

File systems

Om bestanden naar een disk te kunnen schrijven moet er een soort van database bijgehouden worden met op welke track en sector bij het bestand horen, maar ook gegevens als van wie is het bestand en wie heeft er toegang tot het bestand. Al deze gegevens worden bijgehouden door het file system.

In de loop der tijd zijn er verschillende bestandssystemen bedacht voorbeelden zijn FAT, ExtFAT, FAT32, NTFS, ext3, ext4, HFS+ en nog vele anderen. Sommige besturingssystemen ondersteunen alleen hun eigen bestandssysteem, Linux ondersteunt er heel veel. Je kan in Linux vaak dus schijven van andere OSen lezen en schrijven.

13.1 Formatting file systems

Om een disk voor te bereiden op het ontvangen van data moet de disk eerst voorzien worden van de nodige structuren die uniek zijn voor het specifieke bestandssysteem. Het aanbrengen van deze structuren heet het formateren van de disk. Voor elke bestandssysteem heb je dan ook een eigen format tool.

De format tool op een Linux systeem heet `mkfs` (make filesystem). Voor de verschillende file systems is er een eigen tool, die begint met mkfs, dan een punt en dan de naam van het bestandssysteem zoals Linux het kent. Voorbeelden: `mkfs.exfat`, `mkfs.ext3`, `mkfs.fat`, `mkfs.msdos`, `mkfs.vfat`.

13.2 Mounten van lokale bestandssystemen

Wat je zal zijn opgevallen als je een Windows-gebruiker bent is dat Linux geen drive letters heeft. In Linux is het hele systeem één groot file system waarbij de verschillende disks of partities gemount worden op een directory in het file system.

```
$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0 931.5G  0 disk
|-sda1   8:1    0 931.5G  0 part /home/dennis
sdb      8:16   1  59.8G  0 disk
|-sdb1   8:17   1  58.8G  0 part /
|-sdb2   8:18   1     1K  0 part
|_sdb5   8:21   1   976M  0 part [SWAP]
```

13.3 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 14

Netwerk configuratie

Hoofdstuk 15

Software installeren

Hoofdstuk 16

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 naast 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.

In het document Linux Introductie (ook beschikbaar via https://github.com/DennisLeeuw/Linux) staat beschreven hoe je van C-source code naar een vorm komt die de CPU begrijpt. Dat proces heet compileren. Je gebruikt een compiler om bron-code om te zetten naar binaire code.

Er is nog een andere manier om een programma te draaien op je computer. Je kan ook de broncode omzetten naar binaire code via een runtime engine. Programmeertalen die hiervan gebruik maken heten scripting talen.

Het belangrijkste verschil tussen de twee vormen is de manier waarop een applicatie verspreid wordt. Als je de broncode compileert en er een binairbestand van maakt dan kan je alleen het binaire bestand delen en iedereen met dezelfde OS-versie en hardware kan jouw software dan gebruiken. Dit is wat bijvoorbeeld Apple en Microsoft doen en hoe veel van ook de Linux software verspreid wordt via de repositories. Als je software een script is dan is de verspreiding per definitie als broncode. Een shell-script, JavaScript, PHP of Python wordt bijna altijd als broncode verspreid en een runtime engine wordt gebruikt om het script te starten op de computer.

Scripting had de naam om traag te zijn, maar dat is bij moderne scripting talen zoals Python inmiddels bijna niet meer het geval.

We kunnen programeertalen grofweg opdelen in twee soorten:

- Procedural Programming

- Object Oriented Programming (OOP)

Een Procedural programming language is gebaseerd op het gebruik van procedure aanroepen (procedure calls). Een Procedure is een routine of subroutine, misschien beter bekent als functies. Een functie of routine is een blok met commando's die bij elkaar horen en die vanuit het hoofdprogramma één of meer keren aangeroepen kan worden.

Een Object Oriented language is een taal die is gebaseerd op Objecten. Een object is programma-code met data. Waar een functie alleen de programma code bevat bevat een object ook de data. De code bestaat net als bij Procedural languages uit procedures, maar heten dan methods en de data kan aangesproken worden als attributes of fields (velden). Er is gebleken dat veel vertalingen van functies in de wereld die we willen automatiseren zich makkelijker laten vertalen in objecten.

16.1 C

C is een procedurele programmeertaal. Hij is ontworpen bijna gelijk met het ontstaan van Unix. De ontwerpers van C zijn Dennis Ritchie en Brian Kernighan, de taal stamt al uit 1969 en zijn voorlopers waren daadwerkelijk de talen 'A' en 'B'.

Een voorbeeld van een programma geschreven in C is Hello World uit het C programmeer boek. Dit is het eerste programma dat je als voorbeeld kreeg. Dit heeft veel navolging gehad in andere talen. Wij zullen bij elke taal die we hier noemen een voorbeeld geven van Hello World.

```
#include <stdio.h>

int main() {
        printf("Hello, World!");
        return 0;
}
```

Voor C heb je een compiler nodig om de broncode om te zetten in iets dat de computer begrijpt:

```
$ gcc hello.c
$ ./a.out
```

16.2 C++

C++ is een Object Oriented language. C++ heeft een compiler nodig om van de broncode iets te maken dat de computer begrijpt.

Een voorbeeld van het Hello World programma in C++:

```
#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```

Voor de vertaling maar machinetaal gebruiken we `gcc`:

```
$ gcc helloworld.cpp
$ ./a.out
```

16.3 Perl

Perl is begonnen als procedurele taal die voornamelijk gebruikt werd voor het verwerken van data. Later is er ook de mogelijkheid toegevoegd om object georienteerd te kunnen werken. Perl is een scripting taal die ook hoofdzakelijk zo gebruikt wordt.

```
print "Hello, World!";
```

We gebruiken `perl` als runtime engine om het perl-script te draaien:

```
$ perl helloworld.pl
```

16.4 PHP

PHP is **de** programmeer taal van het Internet (geworden). PHP draait op de webserver en genereert de output die door webservers aan de aanvrager gegeven wordt. Je kan PHP echter ook op de commandline gebruiken zonder web output.

Een PHP-script dat je op de commandline kan gebruiken:

```
<?php
  echo "Hello, World!";
?>
```

Om het op te starten hebben we de PHP runtime engine nodig:

```
$ php helloworld.php
```

Maar PHP is eigenlijk bedoelt voor op het web, dat betekent dat er een webpagina moet komen met daarop Hello, World!:

```
<?php
    echo "<html>";
    echo "<head>";
    echo "<title>PHP Hello World</title>";
    echo "</head>";
    echo "<body>";
    echo "<p>Hello, World!</p>";
    echo "</body>";
    echo "</html>";
?>
```

Je kan dat ook op de commandline testen:

```
$ php helloworld_web.php
```

16.5 Python

Python is een Object Oriented scripting taal.

Voorbeeld van een Python:

```
print ('Hello, World!')
```

We gebruiken de `python` of `python3` runtime engine om het script te draaien:

```
$ python3 helloworld.py
```

16.6 Java

Java is een object oriented taal die gebruikt kan worden als gecompileerde applicatie of als scripting taal. Op de meeste linux distributies is Java niet standaard geïnstalleerd. Installeer de Oracle versie of OpenJDK als je de volgende oefeningen wil kunnen doen.

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Je kan direct de java runtime engine gebruiken:

```
$ java helloworld.java
```

Om java te kunnen compileren heb je de `javac` compiler nodig:

```
$ javac helloworld.java
```

Hoofdstuk 17

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.

17.1 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.

17.2 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 kan meegeven zodat je kan laten weten dat het script goed (0) of fout (1) geëindigd is. Doe je dit niet dan is de exit code de error-code van het laatst uitgevoerde commando.

Voer het script uit en geef direct daarna het commando

```
$ echo $?
```

als je nu de regel met daarop `exit 0` vervangt door `exit 1` en dan weer het script draait en het commando

```
$ echo $?
```

geeft dan zie welke invloed het exit commando heeft.

17.3 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
```

17.4 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.

17.5 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 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
```

17.6 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 Controlleer of integers gelijk zijn (equal)

-ne Controlleer of integers ongelijk zijn (not equal)

-gt Controlleer of eerste integer groter is dan de tweede (greater then)

-ge Controlleer of de eerste integer groter of gelijk is aan de tweede (greater or equal)

-lt Controlleer of eerste integer kleiner is dan de tweede (less then)

-le Controlleer 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 het script een keer door de `if` loopt, een keer door de `elif` en een keer door de `else`.

17.6.1 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`.

17.7 for

Om een handeling een bepaald aantal keren uit te voeren kunnen we gebruik maken van `for`. Met behulp van `for` kunnen we een loop doorlopen. Het `for` commando heeft twee hulpcommando's nodig, namelijk `do` en `done`. Na het `for` commando komt de conditie waaraan we moeten voldoen en tussen `do` en `done` staat de loop die doorlopen wordt. Laten we beginnen met een voorbeeld:

```
#!/bin/bash
# (C) 2021 Dennis Leeuw
# License: GPLv3
# Tel van 1 tot 10

list="1 2 3 4 5 6 7 8 9 10"

for i in ${list}
do
        echo $i
done

#END
```

In dit script wordt op basis van een lijstje getallen elk getal één voor één aan de variabele i toegekend, dit is de conditie waaraan de `for` loop moet voldoen. Zolang er nog een parameter (getal) is wordt de loop tussen `do` en `done` doorlopen.

Tussen de commando's die het begin (`do`) en het einde van de loop (`done`) aangeven staan de commando's die in de loop uitgevoerd moeten worden. In ons geval is dat maar één commando namelijke echo $i.

Wat je misschien opgevallen is is dat de shell automatische weet dat hij de getallen 1 tot en met 10 als losse waarden moet beschouwen en deze aan de variabele i moet toekennen. We zouden het script kunnen herschrijven met quotes en zien wat er dan gebeurd:

```
#!/bin/bash
# (C) 2021 Dennis Leeuw
# License: GPLv3
# Tel van 1 tot 10

list="'1 2' 3 4 '5 6' 7 8 '9 10'"

for i in ${list}
do
        echo $i
done
```

```
#END
```

Neem dit script over, let goed op de enkele quotes, en start het script. Ook hier geldt dus wat de enkele quotes (ticks) ervoor zorgen dat spaties niet meer gebruikt worden als scheidingsteken.

De vraag die hopelijk nu ontstaan is is waarom gebruikt de shell een spatie als scheidingsteken? Dat komt omdat de shell een interne variabele kent die `IFS` heet en dat staat voor Internal Field Separator (interne veld scheider). Deze variabele bepaalt hoe de shell met velden omgaat. Standaard gebruikt de shell, spaties, tabs en de nieuwe regel (enter) als scheidingstekens. Ik kan het bovenstaande script dan ook herschrijven met nieuwe regels zonder dat de werking veranderd:

```
#!/bin/bash
# (C) 2021 Dennis Leeuw
# License: GPLv3
# Tel van 1 tot 10

list="1
2
3 4
5
6 7
8
9 10"

for i in ${list}
do
        echo $i
done

#END
```

Het is minder goed leesbaar, maar voor de werking maakt het niet uit. We kunnen de IFS ook aanpassen. We maken nu een nieuw script waarin we het scheidingsteken (`IFS`) naar een newline (nieuwe regel) zetten en de `list` variabele ophakken in twee stukken:

```
#!/bin/bash
# (C) 2021 Dennis Leeuw
# License: GPLv3
# Tel van 1 tot 10

# Set the IFS variable to newline
IFS=$'\n'

list="1 2 3 4 5
6 7 8 9 10"
```

```
# Count from 1 to 10 ... or not
for i in ${list}
do
        echo $i
done

#END
```

De uitvoer zal er dan zo uitzien

```
$ ./script_for_IFS.sh
1 2 3 4 5
6 7 8 9 10
```

Alleen de newline scheidt dan nog de delen en de spaties worden niet meer gezien als scheidingsteken.

Je kan de `IFS` naar elk willekeurig character zetten:

```
#!/bin/bash

IFS="n"
zin="Dit is een zin en daarin komt vaak de n voor"
for i in $zin
do
        echo $i
done
```

De n is nu het scheidingsteken geworden en dat betekent dat de uitvoer van dit script er zo uit komt te zien:

```
$ ./testscriptIFS.sh
Dit is ee
 zi
 e
 daari
 komt vaak de
 voor
```

Je ziet dat alle n-en zijn verdwenen en dat de zin is opgehakt op de plek waar de n stond.

17.7.1 Opdracht: Werken met for

Voor elke nieuwe gebruiker willen we vast een aantal directories maken binnen het bedrijf waarin we werken. Als de directory al aangemaakt is moet die niet gemaakt worden en als hij nog niet bestaat moet hij wel aangemaakt worden. De input van het script moet de naam van de gebruiker zijn. Maak gebruik

van `for` om in een loop de directories aan te maken. De directory-tree die aangemaakt moet worden is:

- Documenten/Werkoverleg
- Documenten/Werkoverleg/Verslagen
- Documenten/Werkoverleg/Actiepunten
- Documenten/Handleidingen
- Documenten/Klanten
- Documenten/Klanten/Systeem_documentatie
- Documenten/Servers/Systeem_documentatie
- Documenten/Servers/Inkoop_documenten
- Documenten/Servers/Handleidingen
- Scripts

17.8 while

Om door een lijst met elementen te lopen tot een bepaalde conditie is bereikt gebruiken we `while`. `while` kent hetzelfde begin en eind van de loop als `for`, namelijk `do` en `done`. Een voorbeeld van het gebruik van `while` zou kunnen zijn:

```
#!/bin/bash
# (C) 2021 Dennis Leeuw
# While voorbeeld

i=1
while [ $i -le 10 ]
do
        echo $i
        i=$(($i+1))
done

#END
```

Dit script lijkt erg op de `for`-loop uit de vorige paragraaf. We vertellen hier dat de variabele `i` 1 is en vragen aan `while` om de loop te doorlopen zolang `i` kleiner of gelijk is aan 10. Om te zorgen dat `i` bij elke keer dat de loop

doorlopen wordt 1 hoger wordt, moeten we bij `i` 1 optellen. Rekenen in de shell doen we door gebruikt te maken van dubbele haken, de uitkomst van de berekening zetten we weer in `i` waardoor `i` 1 hoger wordt. Deze manier van rekenen kan je ook op de commandline gebruiken, daar heb je niet alleen de shell voor nodig.

In de opgave uit de vorige paragraaf heb je een script gemaakt dat bepaalde directories aanmaakt op basis van een gebruikersnaam die wordt ingegeven door de gebruiker. Van de Linux commando's ben je inmiddels gewend dat je met opties waarden kan meegeven. Het zou natuurlijk heel mooi zijn als je die gebruikersnaam ook met een optie mee zou kunnen geven aan een script. En dat kan ook. In scripting heten parameters of opties die je meegeeft bij het opstarten van een script positional parameters. Deze positional parameters hebben nummers en lopen van 0 tot het getal van de laatste parameter. Positional parameter 0 is altijd de naam van het script inclusief het pad waarmee het is opgestart.

```
#!/bin/bash
# (C) 2021 Dennis Leeuw
# Positional parameters

echo "Dit script is opgestart als: $0"

echo "Parameter 1 is: $1"
echo "Parameter 2 is: $2"
echo "Parameter 3 is: $3"

#END
```

als we dit script opstarten met wat parameters kan krijgen we dit:

```
$ ./scriptWhilePosPar.sh aap noot mies
Dit script is opgestart als: ./scriptWhilePosPar.sh
Parameter 1 is: aap
Parameter 2 is: noot
Parameter 3 is: mies
```

De eerste optie is in $1 terecht gekomen, de tweede in $2 enzovoort.

In dit script is het natuurlijk onhandig dat we voor elke positionele parameter een regel schrijven. Het zou makkelijker zijn als we de positionele paramaters één voor één zouden kunnen doorlopen. Een commando dat handig is bij het gebruik van positional parameters is `shift`. Met `shift` schuiven we de positional parameters een positie op. De waarde van parameter 3 wordt de waarde van parameter 2, de waarde van parameter 2 wordt de waarde van parameter 1 en de waarde van parameter 1 wordt weggegooid. Dit kunnen we gebruiken om zo door de hele lijst aan positional parameters te lopen.

```
#!/bin/bash
# (C) 2021 Dennis Leeuw
# While voorbeeld 2

echo "Dit script is opgestart als: $0"

i=1
while [ "$1" != "" ]
do
        echo "Parameter $i is: $1"
        shift
        i=$(($i+1))
done

#END
```

De uitvoer van dit script is hetzelfde als het vorige, maar we hoeven nu niet meer voor elke parameter een regel te maken, plus dat het ook niet meer uitmaakt hoeveel parameters we meegeven. Met het gebruik van `shift` en `while` kunnen we de positionele parameters doorlopen opzoek naar opties die meegegeven zijn aan een script.

17.8.1 Opdracht: Werken met While

Maak een script dat werkt met de volgende 3 opties:

- Als er een optie -u meegegeven wordt, dan is wat er achter -u komt een gebruikersnaam
- Als er een optie -d meegegeven wordt, dan is wat er achter -d komt een directory
- Als er een optie -h gegegeven wordt dan wordt er een beschrijving van het script weergegeven en stopt het script.

De opties -u en -d mogen in willekeurig volgorde aan het script meegegeven worden. Laat als resultaat op het scherm zien welke gebruikersnaam en welke directory als input gegeven zijn.

17.9 Opdracht: Gemiddelde cijfer van een student

In het klas.csv bestand staan de leerlingen van meester Tom. De meeste leerlingen hebben 5 toetsen gemaakt, maar een enkeling door ziekt minder.

Meester Tom wil het gemiddelde cijfer weten per student, zodat hij het eindcijfer met de klas kan delen. Voor degene die een toets gemist hebben wordt het cijfer niet door 5 gedeeld maar door het aantal toetsen dat ze echt gemaakt hebben.

Index