

TIRLNX01

Keuzevak Linux

Dictaat

Auteurs:

KEVIN VAN DER VLIST
kevin@kevinvandervlist.nl

Gedoceerd te:

en

HOGESCHOOL ROTTERDAM
Vestiging Academieplein

PAUL SOHIER
paul@paulsohier.nl

Versie 1.2

6 december 2011

Samenvatting

We proberen jullie wegwijs te maken binnen een Linux omgeving.
Hopelijk slagen we hierin.

“UNIX IS USER-FRIENDLY. IT’S JUST VERY SELECTIVE ABOUT WHO ITS
FRIENDS ARE.”

Voorwoord

De structuur van dit dictaat is gebaseerd op het dictaat *Linux Starter Pack*[4], geschreven door Remko de Vrijer en Peter Berends. We hebben de structuur als basis gebruikt, en verder het document helemaal herschreven met up to date informatie. Daarnaast is het document ook uitgebreid met extra informatie. Enkele kleine delen van de tekst, voornamelijk van technieken die momenteel weinig gebruikt worden, zullen nog uit het originele document komen.

Inhoudsopgave

Voorwoord	1
Inhoudsopgave	11
Inleiding	12
1 Introductie	14
1.1 De geschiedenis	14
1.2 De GNU	15
1.3 Niet vrije software?	15
1.4 Ik wil mijn eigen distributie maken?	16
1.5 De kracht van Linux	16
2 Linux en Hardware	18
2.1 Harde schijven	18
2.1.1 eSATA	18
2.2 USB	19
2.2.1 USB-sticks	19
2.2.2 Muizen en toetsenborden	19
2.3 Netwerkkarten	19
2.4 Overige hardware	20
3 Installatie	21

3.1	Voordat we beginnen	21
3.2	Installatie starten	22
3.3	fdisk	23
3.4	De installatie	24
3.4.1	Linux partities	25
3.5	Packages	26
3.6	LILO	30
3.7	Laatste configuratie	32
3.8	GRUB installatie	32
3.8.1	Downloaden en installeren GRUB	32
3.8.2	grubconfig	33
4	Het systeem	34
4.1	root	34
4.1.1	root worden	35
4.2	Reboot, Shutdown & Runlevels	35
4.3	Aanmaken van gebruikers	36
4.4	Terminals	37
4.5	Mountpoint	37
4.5.1	Devpts	38
4.5.2	Proc	38
4.5.3	Tmpfs	39
4.5.4	Swap	39
4.5.5	Mount	39
4.6	Slackware Package	40
4.7	De man pages	40
4.8	Libraries	41
4.8.1	Static libraries	41
4.8.2	Shared libraries	42

ldconfig	42
5 Inrichting	44
5.1 Inleiding	44
5.2 Linux Standard Base	45
5.3 File System Hiërarchie	45
6 Editors	48
6.1 Pico	48
6.1.1 Bediening	49
6.2 VI(M)	49
6.3 Emacs	50
7 User management	51
7.1 Commando's	51
7.1.1 who	51
7.1.2 w	51
7.1.3 last	52
7.1.4 getent	52
7.1.5 usermod	52
7.1.6 chown	52
7.2 User Identifiers	53
7.2.1 Real User ID	53
7.2.2 Effective User ID	53
7.3 File mode bits	54
7.3.1 Waardes	56
7.3.2 Gebruik van numerieke waarden	56
7.3.3 umask	56
8 Proces management	58

8.1	Processen bekijken	58
8.1.1	UID	59
8.1.2	PID	59
8.1.3	PPID	59
8.1.4	C	59
8.1.5	STIME	59
8.1.6	TTY	60
8.1.7	TIME	60
8.1.8	CMD	60
8.2	Termination	60
8.3	Zombies	61
8.4	Load	61
9	Linux networking	63
9.1	Configuratie	63
9.1.1	netconfig	63
9.1.2	Handmatige configuratie	63
9.1.3	/etc/hosts	64
9.1.4	/etc/resolv.conf	64
9.1.5	ifconfig	64
9.2	SSH	65
9.2.1	Het verleden	65
9.2.2	OpenSSH	66
9.2.3	De eerste keer	66
9.2.4	Verbinding	67
9.2.5	Problemen	67
9.3	SCP	67
9.4	FTP	68
9.5	SFTP	68

10 X	70
10.1 Geschiedenis	70
10.2 Implementatie	70
10.2.1 X server	71
10.2.2 X client	71
10.2.3 Protocol	71
10.2.4 Communicatie	71
10.3 Window Manager	72
10.4 Installatie	72
10.4.1 KDE	73
10.4.2 Fluxbox	73
10.5 X configureren	73
10.6 X Starten	74
10.7 Remote X	74
11 Device files	75
11.1 Device types	76
11.1.1 Character devices	76
11.1.2 Block devices	76
11.2 Belangrijke files	76
11.2.1 null	77
11.2.2 zero	77
11.2.3 random	77
11.2.4 urandom	77
12 Basiscommando's	78
12.1 awk	78
12.2 bzip2	79
12.3 cat	79
12.4 cron	79

12.5 diff	80
12.6 file	81
12.7 find	81
12.8 fsck	82
12.9 grep	82
12.10gzip	82
12.11ldd	83
12.12less	83
12.13ln	83
12.14man	84
12.15more	84
12.16nice	84
12.17patch	85
12.18sed	85
12.19stat	85
12.20tail	86
12.21tar	86
12.22wget	86
13 Shell	88
13.1 Inleiding shell	88
13.2 De theorie	90
13.2.1 Interactive command line en scripting program language?	91
13.2.2 Shebang	91
13.2.3 Dash in Slackware?	92
13.2.4 In en output	93
13.2.5 Pipeline	93
STDERR redirecten	94

13.2.6	TAB-completion	95
13.2.7	Opdrachtalliassing	95
13.2.8	Historylist	95
13.2.9	Jobcontrol	96
13.2.10	Patronen, Joker-tekenen en Accolades	97
13.2.11	Opdracht-substitutie	98
14	Shell scripting	99
14.1	Statements, expressies en controle structuren	99
14.2	Variabelen en het if-statement	99
14.3	Systeem variabelen	100
14.4	Expressies	101
14.4.1	String expressies	101
14.4.2	Integer expressies	101
14.4.3	Bestands expressies	102
14.4.4	Logische operatoren	102
14.5	De shell parameter variabelen	102
14.6	Het case-statement	103
14.6.1	Case voorbeeld	104
14.6.2	HELP, dit script werkt niet!	104
14.7	Het while statement	105
14.8	Het untill-Statement	105
14.9	Het for-statement	106
14.10	Functies	106
14.10.1	Parameters	107
14.10.2	Shell parameters en functie parameters	108
14.11	Arrays	109
14.11.1	Assignment	109
14.11.2	Benaderen	109

14.12	En nu?	110
14.13	Commando gebruik	110
14.14	Alternatieve scripting mogelijkheden	110
A	Slackware installeren op VirtualBox	112
A.1	Problemen met netwerk/SSH	112
B	BIOS	114
B.1	Boot sequence	114
C	Partities en fdisk	115
C.1	Fdisk	115
C.2	Swap	119
D	Boot manager	120
D.1	MBR	120
D.2	Bootable partitie	120
D.3	GRUB	121
D.3.1	GRUB installeren	122
D.3.2	GRUB boot wijzigen	122
D.4	LILO	123
D.4.1	Installatie	123
D.4.2	Aanpassen LILO configuratie	123
E	Rescue	125
E.1	Nieuwe harddisk	125
E.2	Root password gecompromitteerd	126
E.3	Root password vergeten	127
E.4	Windows geïnstalleerd - Linux weg	127
F	Slackware Packages	128

F.1	pkgtool	128
F.2	installpkg	128
F.3	upgradepkg	129
F.4	removepkg	129
G	Source installatie	130
G.1	Achtergrond	130
G.1.1	Voordelen	130
G.1.2	Nadelen	131
G.2	Hoe gaan we te werk	131
G.2.1	Source download	131
G.2.2	configure	132
G.2.3	make	133
G.2.4	make install	133
G.2.5	Source management	134
H	SSH	135
H.1	Verbinding opzetten	135
H.2	Authenticatie	136
H.2.1	Public Key Authentication	136
H.2.2	Password authenticatie	136
H.3	Account check	136
H.3.1	Locked	137
H.3.2	DenyUsers	137
H.3.3	DenyGroups	137
H.4	Sessie voorbereiden	137
H.5	Shell	138
H.6	Disconnect	138
I	GNU Emacs	139

I.1	Introductie	139
I.2	Voordelen	139
I.3	Shortcuts	140
I.4	Navigatie in Emacs	140
I.4.1	Configuratie	141
J	VIMProved	143
J.1	Navigatie in VIM	143
J.2	~/vimrc	144
K	General Public License v2	145
K.1	Uitwerking	145
K.1.1	Rechten	145
K.1.2	Plichten	146
K.1.3	Verbod	147
K.1.4	Opmerkingen	147
K.2	Licentie	148
	Bibliografie	156
	Index	157

Inleiding

Dit dictaat is erop gericht om jullie op een goede manier in te leiden binnen een *GNU/Linux* systeem. We zullen dit echter op een andere manier aanpakken dan men in eerste instantie zou denken.

Om een duidelijk beeld te creëren zonder verschillende termen door elkaar te halen zullen door het dictaat de volgende drukstijlen gebruikt worden.

1. Normale tekst is gewoon uitleg.
2. “Tekst in quotes” zal betekenen dat we dingen extra willen benadrukken. Denk aan gebruikersnamen en menu-items.
3. *Schuin gedrukte woorden* zullen begrippen of namen zijn.
4. Commando's zullen in `monospaced font` afgedrukt zijn.
5. Voorbeelden met code of de uitvoer van commando's zullen in blokken verschijnen met regelnummers. Regelnummers zijn geen onderdeel van de code.

1| Dit is een voorbeeld. |

Als eerste zal er duidelijk gemaakt moeten worden dat er niet op een “grafische” manier naar een systeem gekeken zal worden. Er zal namelijk geen gebruik gemaakt worden van een *GUI*. De reden hiervoor is de drang om een systeem “te verbergen” voor gebruikers. Deze krijgen hierdoor geen mogelijkheid om eens goed onder de motorkap te kijken.

Ook zal er in server situaties geen *GUI* gebruikt worden. Waarom zou iemand de kostbare resources van een server willen verspillen aan rekentijd voor een grafische omgeving? Dit geldt alleen maar sterker als de snelheid van een geoefende *shell* gebruiker in ogenschouw wordt genomen.

Dit dictaat zal worden gericht op de distributie *Slackware*. Hier is voor gekozen omdat het relatief weinig wordt gebruikt en omdat het een puristische

instelling heeft met betrekking tot de inrichting van het systeem. Dit resulteert in meer “handwerk” om het systeem te configureren, maar laat wel beter zien hoe het nu precies in elkaar zit.

We hopen dat dit dictaat jullie zal helpen bij het begrijpen van de *GNU/Linux* wereld.

Hoofdstuk 1

Introductie

Linux is net als *Windows* een operating system, het “programma” wat draait op de computer en waarmee de computer bestuurd wordt. *Linux* is ooit bedoeld geweest voor gebruik op een server en wordt daar ook op dit moment nog primair voor gebruikt. Ook als vervanger voor *Windows*, met een grafische schil is *Linux* tegenwoordig goed geschikt. Wel is er nog duidelijk te zien dat *Linux* primair bedoeld is voor gebruik op een server. Niet voor niets draait *Linux* op de meeste web servers. De *GUI* van *Linux* is ook tegenwoordig goed bruikbaar voor normaal gebruik en er zijn dus ook steeds meer mensen die *Linux* gebruiken voor hun “normale” computer. In dit dictaat gaan we geen gebruik maken van een *GUI*, maar gaan we puur de *Shell* gebruiken. Met een *GUI* kan je vrij eenvoudig alles instellen en wijzigen, terwijl je met alleen een *shell* je veel meer zelf moet doen.

1.1 De geschiedenis

In 1991 wilde de Finse student Linus Torvalds gebruik maken van *UNIX*. Omdat *UNIX* echter een betaald operating system was, ging hij aan de slag met *Minix*. Dit voldeed al snel niet meer aan de eisen welke hij stelde en hij besloot een eigen kernel te gaan schrijven. Eigenlijk wilde hij een compleet operating system gaan maken, maar hij schreef uiteindelijk alleen de kernel. Naast de zelf geschreven kernel gebruikte hij de *GNU tools*. Vandaar dat wanneer er tegenwoordig *Linux* word gezegd, er de *GNU/Linux* combinatie bedoeld wordt. *Linux* is de kernel, met daarop de *GNU tools* in *userland*.

De eerste versie was niet echt bruikbaar voor de gemiddelde mens, het was het meer een speeltje voor programmeurs en hackers. Door het open karakter en de actieve gebruikers gemeenschap is hier toch een stabiel en volledig werkend operating system uit voortgekomen. In 1994 kwam de eerste versie

dan echt uit en sindsdien is de kernel ook sterk verbeterd. Tegenwoordig draait *Linux* niet alleen op standaard computers, maar ook op *embedded systemen*, zoals routers en *telefoons*. In de toekomst zal dit alleen maar toenemen. Er zijn momenteel zelfs al koelkasten in de handel die een *Linux* versie draaien.

De kernel is uiteraard ook op dit moment nog steeds in ontwikkeling. Onder leiding van Linus Torvalds worden er nog steeds nieuwe technieken geïmplementeerd en fouten verbeterd. Uiteraard doet hij dit niet in zijn ééntje, iedereen die wilt kan zijn bijdrage doen via de git repository van de kernel. Het aantal personen dat heeft gewerkt aan de source code van de *Linux* kernel bedraagt vele duizenden.

1.2 De GNU

De *GNU foundation* is opgericht door Richard Stallman. Hij is het project begonnen om een vrije implementatie van een *UNIX* systeem te creëren. Het bekendste werk van de *GNU foundation* is de *GNU userland*, het gedeelte wat *Linux* bruikbaar maakt als systeem. Het biedt namelijk een vrije verzameling van zogenaamde *userland utilities*, waardoor het systeem daadwerkelijk gebruikt kan worden. De *GNU foundation* zet zich op allerlei manieren in om te garanderen dat deze software vrij blijft. Naast de software is er ook de *GPL*. De *GPL* is de licentie welke is geschreven door de *GNU foundation*. Alle software welke wordt uitgebracht door de *GNU* wordt vrijgegeven onder de *GPL* license. In bijlage K kan er een samenvatting gevonden worden over de *GPL* en wat de *GPL* precies is. In principe zijn alle *Linux* varianten uitgekomen onder de *GPL*. Vroeger was alle software gebruikt onder *Linux GNU* software, maar tegenwoordig wordt er steeds vaker gebruik gemaakt van software welke als vervanger is geschreven voor de *GNU* variant. Deze vervangers zijn in sommige gevallen ook niet onder de *GPL* uitgegeven, maar onder een andere open source license.

1.3 Niet vrije software?

Uiteraard is er ook voor *Linux* niet vrije software beschikbaar. Sommige distributies steunen dit en hebben via hun package systeem deze niet vrije software gewoon beschikbaar. Maar er zijn ook distributies welke hier zeer strikt in zijn. Een goed voorbeeld hierbij is *Debian* (officieel *Debian GNU/Linux*). *Debian* heeft als doel om enkel en alleen vrije software te gebruiken, dus vrijgegeven onder een open source licentie, zoals de *GPL*. Closed source software, maar ook software welke niet voldoende copyright verleend aan

de originele maker, zal in principe niet verspreid worden via het package system. *Debian* gaat met dit principe erg ver, er wordt zelfs software verwijderd uit de repository van *Debian* als het niet voldoet aan deze eisen¹². Andere distributies zijn hier soms minder streng in.

1.4 Ik wil mijn eigen distributie maken?

Door het vrije karakter is dit een van de mogelijkheden. Veel distributies zijn afgeleid van een andere distributie. Als je kijkt naar de historie van alle distributies [24] zie je dat een aantal distributies aan het begin begonnen zijn, de belangrijkste (en nog steeds gebruikten/ontwikkelde) hierin zijn: *Debian*, *Slackware* (gebaseerd op *SLS*), *Suse* (gebaseerd op *Slackware*) en *Red Hat*. Deze namen zullen je bijna allemaal redelijk bekend in de oren klinken, want ze zijn allemaal nog erg in trek. Je kan hier ook zien dat *Slackware* op dit moment het langst ontwikkelde (en nog steeds in ontwikkeling zijnde) distributie is.

Vanaf de hierboven genoemde distributies zijn andere distributies voortgekomen, zogenaamde forks (afsplitsing). Ze kunnen op deze manier een geheel eigen distributie maken, zolang deze maar voldoet aan de voorwaarde waar het origineel onder is uitgebracht (dit is bijna altijd de GPL).

Je kan dus zelf zo van het ene op het andere moment een eigen distributie beginnen, zonder dat je erg veel werk hebt. Uiteraard zal je hiervoor heel wat meer kennis nodig hebben als uit dit document, maar het bevindt zich onder de mogelijkheden.

1.5 De kracht van Linux

De kracht van Linux is natuurlijk het open karakter. Iedereen kan, als die persoon zich hiervoor openstelt, het aanpassen naar zijn eigen wens. Mochten er dus aanpassingen nodig zijn, zal er niemand zijn die de programmeur tegenhoudt om dit door te voeren.

Wanneer een wijziging ook voor anderen van toegevoegde waarde is, kan *upstream* (het ontwikkelteam van het project) besluiten de verbeteringen te integreren in de software, waarna het ook voor anderen beschikbaar gesteld word. Dit effect treed vooral op wanneer er fouten of veiligheidsrisico's

¹Het Firefox logo en de naam Firefox is een merkenrecht, waardoor het niet voldoet aan de richtlijnen voor vrije software. *Debian* besloot hierop een versie zonder dit merkenrecht te verspreiden. Je zal in *Debian* dus gebruik maken van Iceweasel (en Icedove voor Thunderbird) in plaats van Firefox.

²Zie bijvoorbeeld ook hier <http://www.debian.org/News/2010/20101215>

optreden. Eén willekeurige ontwikkelaar hoeft maar een patch te maken die de problemen verhelpt, waarna het probleem wereldwijd is opgelost. Er is nu geen afhankelijkheid meer van een (onwillende) derde partij.

Het nadeel van deze open structuur is de mogelijkheid tot een wildgroei van projecten. Het is meerdere malen voorgekomen dat door een ruzie binnen een project forks ontstaan. Hierdoor komt er op de lange termijn een enorme wildgroei aan projecten, wat het overzicht voor de gebruikers niet duidelijker maakt. Een echte oplossing voor dit systeem is er niet, omdat de software compleet vrij is. Nadelige gevolgen kunnen echter wel worden beperkt door het gebruik van package managers binnen een distributie. Zij zorgen er dan voor dat er een duidelijke hoeveelheid packages is waaruit gekozen kan worden.

Hoofdstuk 2

Linux en Hardware

Een belangrijk deel in *Linux* is de hardware aansturing. Zonder hardware zal een computer in principe niet werken, hij bestaat immers dan niet meer. Hierdoor is er dus veel focus op hardware support binnen *Linux*. Dit complete hoofdstuk gaat over de hardware in een computer.

2.1 Harde schijven

Harde schijven zijn een vrij belangrijk onderdeel binnen je computer. In het verleden werd er verschil gemaakt tussen *IDE* (PATA) en *S-ATA*. In de huidige kernel wordt er gebruik gemaakt van *libata* waardoor alle schijven via dezelfde methode worden afgehandeld. Alle aanwezige schijven zullen een `/dev/sdX` device aangewezen krijgen, zoals bijvoorbeeld `/dev/sda` voor de eerste schijf. Hierbij staat de `X` voor het schijfnummer in de vorm van een letter. Deze letter wordt toegewezen op de volgorde waarop de schijven aanwezig zijn in de computer. Wanneer er partities aanwezig zijn op een harde schijf krijgen deze na `/dev/sdX` een nummer toegewezen, zoals bijvoorbeeld `dev/sda3` voor de derde partitie van de eerste harde schijf.

2.1.1 eSATA

Met de moderne hardware van tegenwoordig zijn er ook veel externe harde schijven. Veel schijven worden via *USB* aangesloten, maar tegenwoordig zijn er ook schijven verkrijgbaar welke via *eSATA* aangestuurd kunnen worden. *eSATA* is op dezelfde standaard gebaseerd als *SATA*, maar dan voor externe schijven. *eSATA* schijven worden ook door *libata* afgehandeld.

2.2 USB

USB is één van de meest gebruikten aansluitingen op een computer om randapparatuur aan te sluiten. *Linux* biedt ondersteuning aan via *libusb* ondersteuning voor een hoop *USB*-apparatuur. We proberen hier de meest gebruikte *USB*-hardware te bespreken, maar we kunnen uiteraard niet alles bespreken.

2.2.1 USB-sticks

Onder *USB-sticks* valt eigenlijk veel meer dan alleen maar een simpele *USB* stick. Eigenlijk valt alles wat op de computer via *USB* aangesloten wordt en een benaderbaar schrijfbaar medium bevat hieronder. Dit betekent bijvoorbeeld een fotocamera, externe harde schijf (Die niet via *eSATA* aangesloten wordt) enzovoort. Deze *devices* worden net als de normale harde schijven via *libata* aangestuurd. Dit betekent dus dat ze een *sdX* device toegewezen krijgen en dat deze hierna via dit device eenvoudige *gemount* kan worden. Doordat *mount* tegenwoordig het filesystem herkent hoeft er geen parameter opgegeven te worden over welk filesystem gebruikt wordt. Door gebruik te maken van *udev* kan het device ook automatische gemount worden. Dit is bijvoorbeeld handig wanneer er gebruik wordt gemaakt van een *GUI*.

2.2.2 Muizen en toetsenborden

Tegenwoordig wordt er steeds meer gebruik gemaakt van muizen en toetsenborden welke ook via *USB* worden aangestuurd. Doordat er een standaard is ontwikkeld voor de communicatie van deze apparaten is het voor de makers ervan heel eenvoudig om een nieuw device te maken. Ook voor gebruikers is het eenvoudig om een nieuw device in gebruik te nemen. Door het soort *USB* device door te geven weet de *kernel* met wat voor soort device hij te doen heeft en kan hij deze direct aanroepen en aansturen. Hierdoor hoeft niet voor ieder nieuw device een aanpassing voor de *kernel* gemaakt te worden. Dit zou niet te doen zijn en je ontwikkeld een enorm probleem voor de ontwikkelaars om dit bij te houden.

2.3 Netwerkkarten

Een netwerkkkaart wordt in bijna alle gevallen automatisch door de kernel herkend indien er in de kernel een module aanwezig is voor de netwerkkkaart. Tegenwoordig zijn voor de meest bekende bedrade netwerkkkaarten drivers

aanwezig. Voor *WIFI* kaarten geldt eigenlijk hetzelfde. Een uitzondering hierop is wanneer de drivers niet open source zijn en je bijvoorbeeld gebruik maakt van een distributie zoals *Debian*. Zie hoofdstuk 9 over netwerken voor meer informatie over het gebruik van het netwerk en de configuratie.

2.4 Overige hardware

Veel hardware wordt eigenlijk direct ondersteund door de *Linux* kernel. De reden hiervoor is eigenlijk vrij simpel. De meeste hardware welke er is maakt gebruik van *USB*. Doordat *libUSB* ontzettend veel randapparatuur standaard ondersteund maken veel fabrikanten gebruik hiervan. Er zijn eigenlijk maar weinig verschillende soorten poorten op een computer/server aanwezig naast de *USB*-poorten. Hierdoor is het vaak ook niet nodig om extra ondersteuning toe te voegen voor apparaten welke niet gebruik maken van *USB* of bijvoorbeeld *PCI-Express*. Ook *PCI-Express* wordt standaard ondersteund door *Linux*.

Hoofdstuk 3

Installatie

In dit complete dictaat is er gekozen voor de *Slackware* distributie. Dit is gedaan omdat het een van de oudste, nog “levende” distributies binnen de *Linux* wereld is, maar ook omdat je in *Slackware* ontzettend veel zelf moet doen.

In dit hoofdstuk zal de lezer geholpen worden met het installeren van *Slackware*, om precies te zijn met versie 13.37. De installatie kan plaatsvinden op een fysieke computer, maar ook het gebruik van *VirtualBox* is erg handig. Zie voor het gebruik van *Virtualbox* bijlage A.

Wanneer er een nieuwe versie van *Slackware* uit is als de gebruikte versie in dit document is het altijd aan te raden deze nieuwe versie te gebruiken.

3.1 Voordat we beginnen

Voordat we beginnen met de installatie zijn we van het volgende uitgegaan bij het schrijven van dit dictaat;

- We gaan er vanuit dat er geen data aanwezig is op de huidige harde schijf.
- We gaan er vanuit dat je gebruikt maakt van virtualbox, met een harde schijf van 25GiB. Maak je gebruik van een andere schijfgrote, zullen de commando's welke we gebruiken niet kloppen met wat jij moet uitvoeren.
- Als we een scherm niet bespreken of uitleggen, kan je gewoon de standaard waarde uit dat scherm aanhouden.

3.2 Installatie starten

Stop de *cd* van *Slackware* in de *cd-rom* speler en start op van *cd-rom*. Via *VirtualBox* zal de *iso* eerste gemount moeten worden voordat de *VM* wordt gestart. Er zal nu het een scherm komen zoals uit figuur 3.1.

```
ISOLINUX 3.84 2009-12-18 ETCD Copyright (C) 1994-2009 H. Peter Anvin et al
Welcome to Slackware version 13.37 (Linux kernel 2.6.37.6)!

If you need to pass extra parameters to the kernel, enter them at the prompt
below after the name of the kernel to boot (huge.s etc). NOTE: If your machine
is not at least a Pentium-Pro, you *must* boot and install with the huge.s
kernel, not the hugemp.s kernel! For older machines, use "huge.s" at the
boot prompt.

In a pinch, you can boot your system from here with a command like:

boot: hugemp.s root=/dev/sda1 rdinit= ro

In the example above, /dev/sda1 is the / Linux partition.

To test your memory with memtest86+, enter memtest on the boot line below.

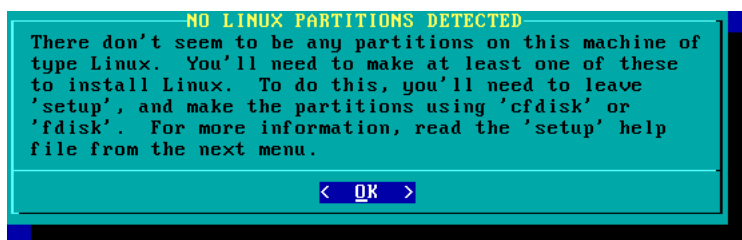
This prompt is just for entering extra parameters. If you don't need to enter
any parameters, hit ENTER to boot the default kernel "hugemp.s" or press [F2]
for a listing of more kernel choices.

boot: _
```

Figuur 3.1: Voor het booten

Hier hoeft niets opgegeven te worden, er kan direct op enter gedrukt worden. Zodra er een vraag wordt gesteld over de toetsenbord layout kan dit aangegeven worden. Waarschijnlijk is het toetsenbord wat gebruikt wordt gewoon *US-INTL*¹ en kan er direct op enter gedrukt worden. Hierna dient er te worden ingelogd als “root”. Type nu *setup*.

Zodra je *setup* hebt gekozen krijg wordt de fout uit figuur 3.2 gegeven. Deze fout komt doordat er nog geen partities gemaakt zijn.



Figuur 3.2: Fout bij het begin

¹Wanneer dit fout ingevuld wordt kan dit later alsnog aangepast worden.

3.3 fdisk

Voordat we dus verder gaan moeten we eerst partities maken. Kies gewoon voor “Ok” en sluit de *setup* af. Hierna kunnen we de partities als volgt aanmaken:

```
1| fdisk /dev/sda
```

Waarbij */dev/sda* de harde schijf is welke gebruikt moet worden voor de *Slackware* installatie. Voer de volgende commando's uit²:

```
1| n
2| p
3| 1
4| <default>
5| +10G
6| n
7| p
8| 2
9| <default>
10| +14G
11| n
12| p
13| 3
14| <default>
15| <default>
16| t
17| 3
18| 82
19| a
20| 1
```

Controleer voordat de tabel wordt weggeschreven of dit klopt. Dit kan met de optie *p*. Het zal overeen moeten komen met het onderstaande voorbeeld³:

```
1| Command (m for help): p
2|
3| Disk /dev/sda: 26.8 GB, 26843545600 bytes
4| 255 heads, 63 sectors/track, 3263 cylinders, total 52428800
   sectors
5| Units = sectors of 1 * 512 = 512 bytes
6| Sector size (logical/physical): 512 bytes / 512 bytes
7| I/O size (minimum/optimal): 512 bytes / 512 bytes
8| Disk identifier: 0x028f6a13
9|
10|   Device Boot      Start         End      Blocks   Id  System
11| /dev/sda1  *           0      20973567    10485760   83  Linux
12| /dev/sda2             20973568   50333695    14680064   83  Linux
```

²Zie bijlage C voor meer informatie over *fdisk* en wat de onderstaande commando's/parameters betekenen.

³Uiteraard er van uitgaande dat de harde schijf van dezelfde grote is. Controleer met de verhoudingen met de door jou gebruikte harde schijf.

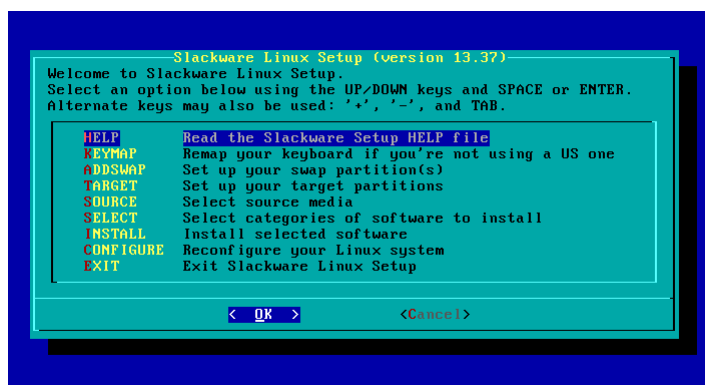
```
13 | /dev/sda3          50333696      52428799      1047552      82  Linux
    | swap
```

Wanneer het klopt kan de tabel met *w* worden weggeschreven naar de schijf. De partitietabel is dan opgeslagen. *Slackware* weet nu welke ruimte hij kan gebruiken. De installatie kan nu opnieuw gestart worden door het commando `setup` uit te voeren.

Tip: Door de toets combinatie “ctrl + page_up” en “ctrl + page_down” kan door de terminal historie buffer gelopen worden

3.4 De installatie

Nu begint de echte installatie van *Slackware*. Het menu uit figuur 3.3 komt nu in beeld.



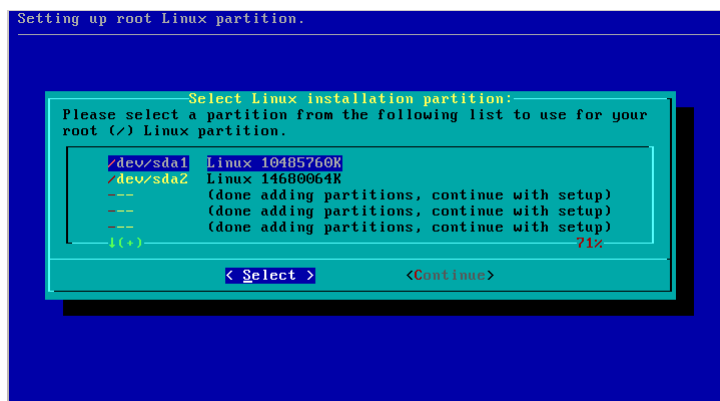
Figuur 3.3: Hoofdmenu van de installatie

De help functie hebben we niet nodig, omdat er al bekend is wat er gedaan moet worden. Het keyboard is ook al geconfigureerd, dus er kan gelijk gekozen worden voor de optie *addswap*. Wanneer het bij *fdisk* goed is geconfigureerd zal hier */dev/sda3* weergegeven worden. Kies hiervoor en druk op Ok.

Er volgt nu een optie om op *bad blocks* te controleren. In een *VM* is dit niet nodig, bij fysieke hardware is het aan te raden om problemen te voorkomen. Hierna zal de *swap* geactiveerd worden.

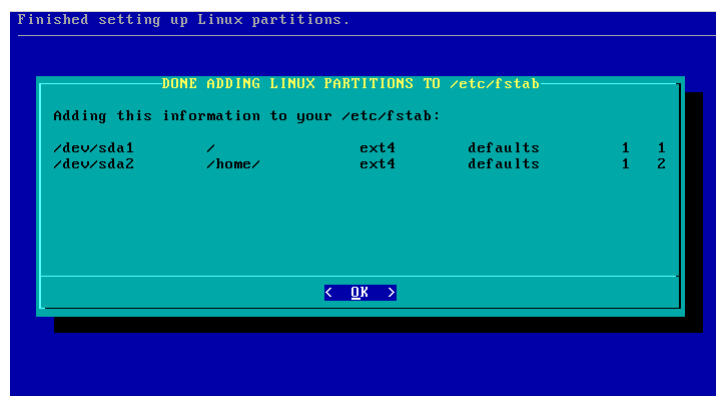
3.4.1 Linux partitions

Er volgt nu een nieuw scherm, zoals te zien in figuur 3.4⁴. *Slackware* wil graag weten op welke partitie de *root* (/) geïnstalleerd kan worden. Kies hier voor */dev/sda1*, de kleinste van de twee.



Figuur 3.4: Partities.

Bij de volgende stap moet het filesystem gekozen worden. Kies voor “Format” en in het volgende scherm het type, bijvoorbeeld *EXT4*⁵. Er kan nu weer een “mount point” worden gekozen. Kies hier voor */home*. Hierna zal er een overzicht te zien zijn, zoals figuur 3.5.



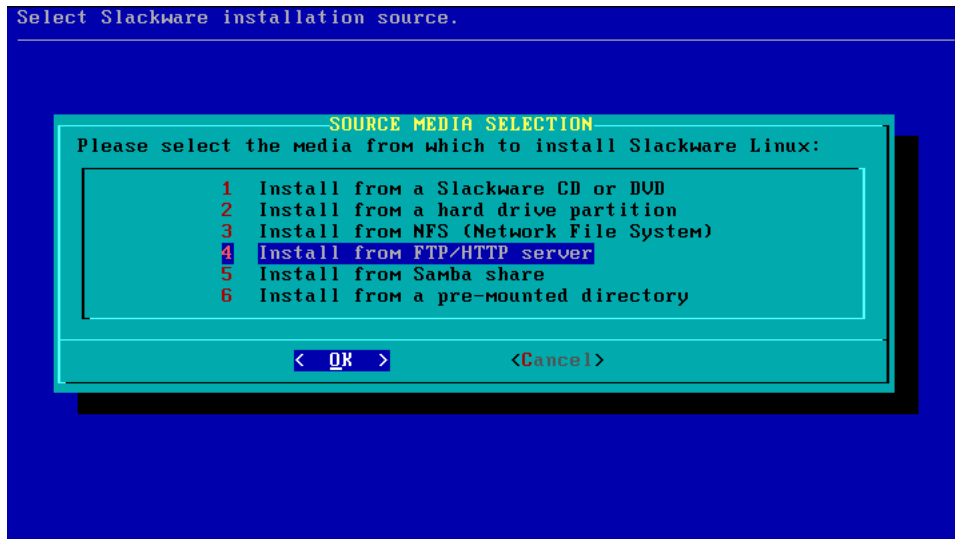
Figuur 3.5: De gemaakte schijfindeling

⁴De waardes kunnen anders zijn, afhankelijk van de eerder gekozen waardes.

⁵Het gaat te ver om in dit document uitgebreid uit te leggen welke type file systems er zijn. *EXT4* werkt in principe correct voor wat wij doen.

3.5 Packages

De harde schijven zijn nu klaar voor gebruik en er kan begonnen met de installatie van het systeem. Zodra er op “Ok” wordt gedrukt zal er gevraagd worden om een locatie van de *packages*. Kies hier voor *FTP/HTTP*⁶. Zie ook figuur 3.6.



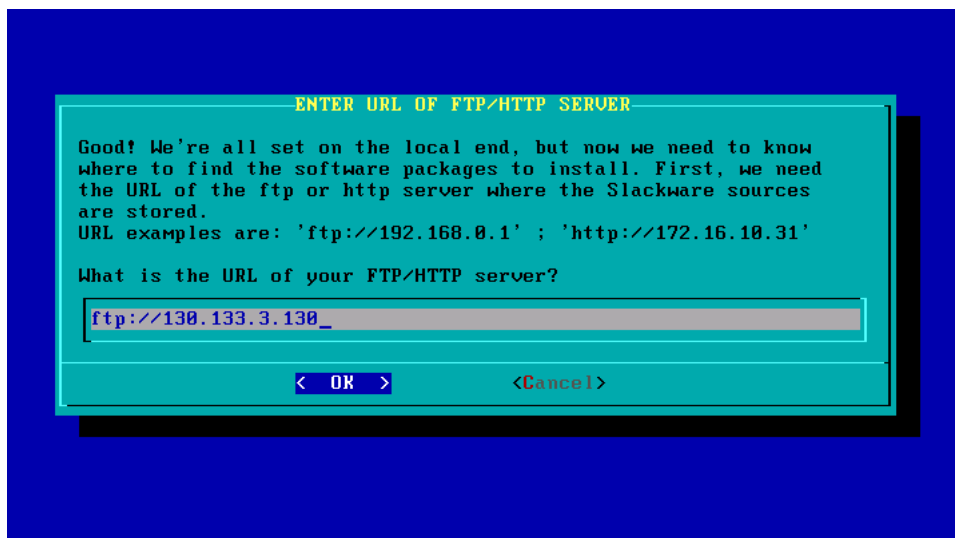
Figuur 3.6: Source media keuze

Er zal nu worden gevraagd naar de netwerk configuratie. Dit is natuurlijk afhankelijk van de situatie. Bij het gebruik van *VirtualBox* zal *DHCP* voldoende zijn. Mocht je geen gebruik maken van *Virtualbox* moet je de gegevens invullen welke voor het lokale netwerk gelden.

Er kan nu een *mirror* worden gekozen. Dit is een plek waar een kopie van alle *Slackware* packages te vinden is. Een voorbeeld kan zijn `ftp://ftp.fu-berlin.de/`⁷, zie ook 3.7. Uiteraard kan ook een andere mirror gekozen worden. Zie hiervoor de *Slackware* site.

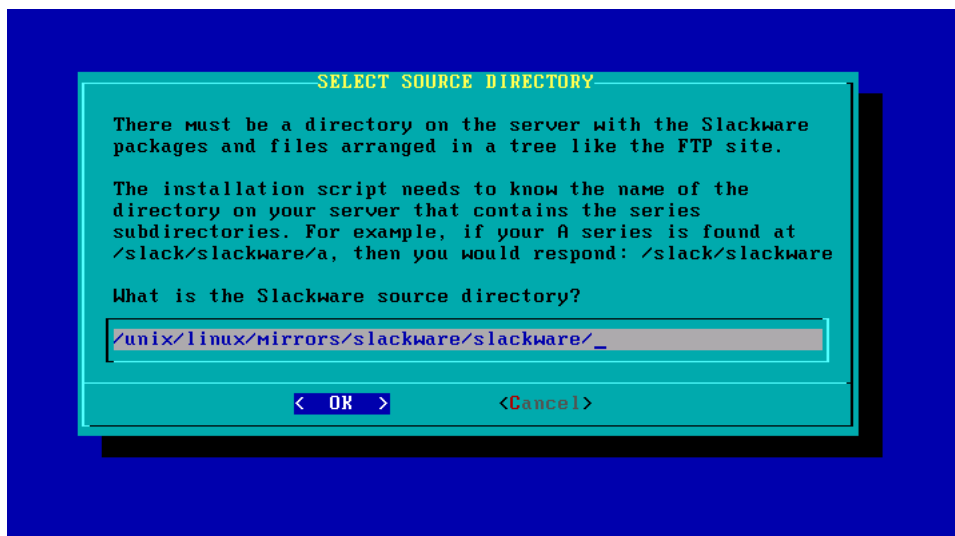
⁶CD is ook een mogelijkheid, alleen vereist het meer cd's

⁷Op de afbeelding is het IP van deze host te zien.



Figuur 3.7: Mirror locatie

Nu zal er ook nog gekozen moeten worden voor een map op de *mirror server*. Op het moment van schrijven is *unix/linux/mirrors/slackware/slackware/* de goede directory. Dit kan bij het gebruik van een andere mirror een andere map zijn, dit is mirror afhankelijk. Zie hierbij ook figuur 3.8.



Figuur 3.8: Mirror locatie

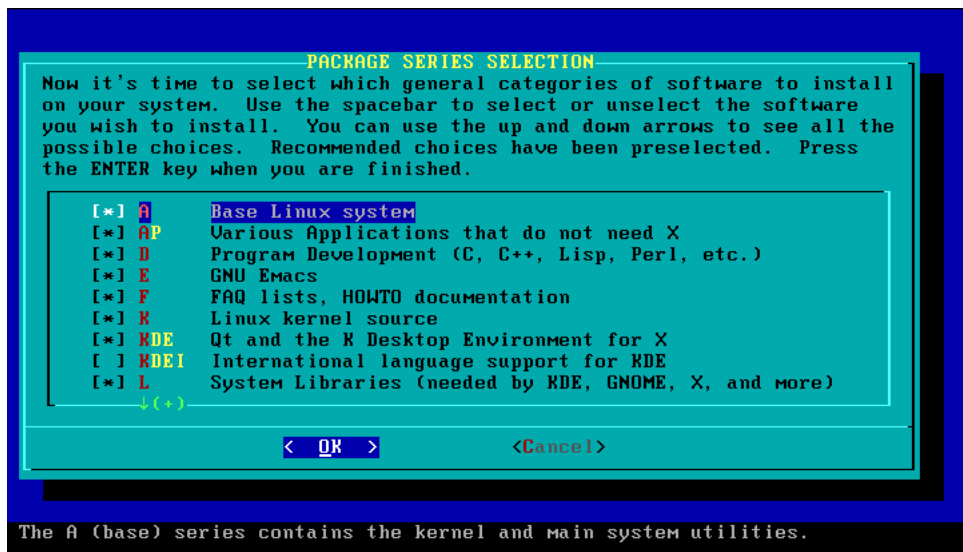
Slackware gaat nu controleren of de mirror gegevens kloppen en je krijgt,

als de gegevens kloppen, de vraag of je het nogmaals wilt proberen of niet. Je kiest hier uiteraard voor niet.

In dit scherm kan er gespecificeerd worden welke packages geïnstalleerd moeten worden. De volgende packages zijn niet nodig, wij willen namelijk geen *GUI* gebruiken. Deselecteer dus de volgende opties:

1. Linux kernel source⁸
2. KDE
3. TEX
4. X
5. XAP
6. Games

Kies nu weer “Ok”, zoals te zien op figuur 3.9.



Figuur 3.9: Packages

Hierna krijg je het scherm zoals in figuur 3.10. Het is aan te raden hier om “full” te kiezen, dit is de snelste optie en hoeft je het minste te doen. Wanneer er een package vereist is om te installeren zal deze nu eerst automatisch van de gekozen *mirror* afgehaald worden. Iedere package zal ook een korte

⁸Deze optie is alleen nodig als er kernels compiled worden op de machine

beschrijving hebben welke op het scherm staat tijdens de installatie. Zodra *Slackware* klaar is met alle packages op te halen en installeren gaat hij vanzelf verder met figuur 3.11. Op deze vraag kies je “skip”.

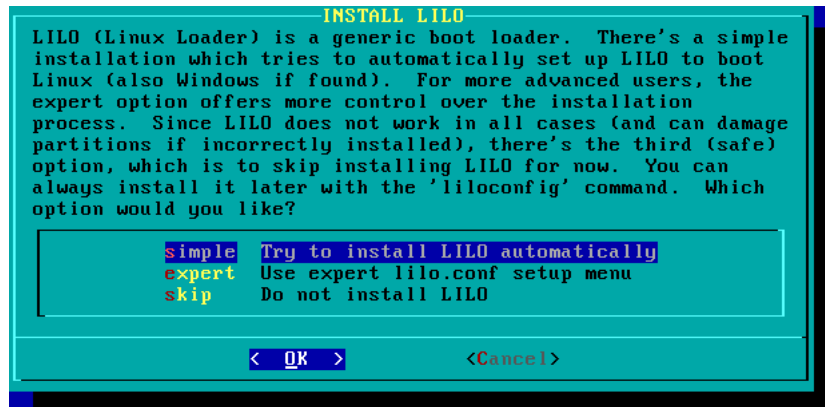


Figuur 3.10: Installatie optie



Figuur 3.11: USB opstart disk

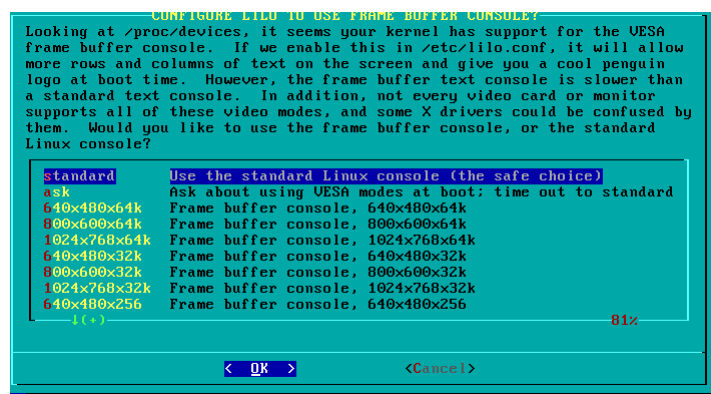
3.6 LILO



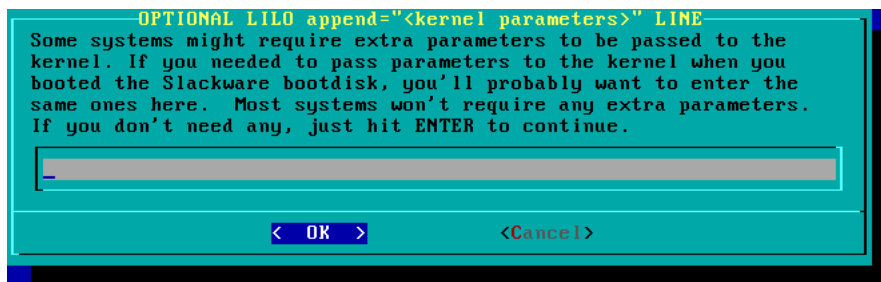
Figuur 3.12: LILO installatie

LIL0 is de standaard bootmanager van *Slackware*, *LIL0* is echter niet heel uitgebreid en je kan er niet veel mee. Hierom hebben wij ervoor gekozen om *GRUB* te gaan gebruiken als bootmanager en gaan we deze later dan ook installeren. Maar voordat we dit kunnen, moeten we eerst een werkend systeem hebben en moet je dus *LIL0* installeren.

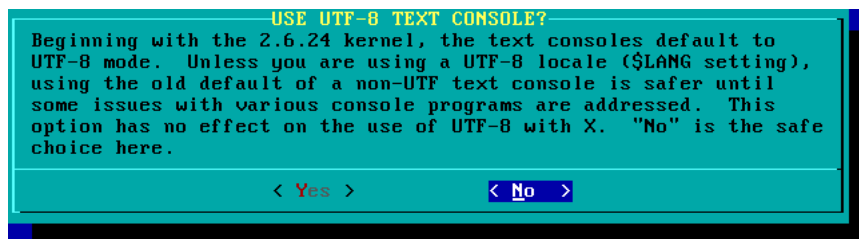
In het scherm van 3.12 moet je “simple” kiezen. Hierna krijg je het scherm uit figuur 3.13. Je kan hier de standaard instelling behouden.



Figuur 3.13: LILO framebuffer



Figuur 3.14: LILO kernel options



Figuur 3.15: LILO UTF8

In sommige gevallen is het nodig om de kernel een aantal extra opties mee te geven om op te starten. Dit kan je in de installatie opgeven in 3.14. In principe hoef je hier niets op te geven en werkt het direct. Voor de echte instellingen krijgen we eerst nog een vraag over UTF8 zoals uit figuur 3.15. Je kan hier gewoon het standaard antwoord kiezen.



Figuur 3.16: LILO locatie

In het scherm van figuur 3.16 wordt gevraagd waar *LILO* geïnstalleerd moet

worden. Je kan hier kiezen voor *mbr*, de standaard keuze. In principe kan je ook kiezen voor *root*, mits je de partitie als *bootable* hebt ingesteld. Als je de instructies correct gevolgd hebt was hij al bootable.

3.7 Laatste configuratie

Kies als muis voor een generieke *PS2* muis. Dit zal de minste kans op problemen geven. Er zal ook nu weer om de netwerk configuratie gevraagd worden. Vul deze weer op de gewenste manier in. Vul als hostname een zelf gekozen naam in.

Op de vraag over welke *services* geactiveerd moeten zijn kan er gewoon met “Ok” de default waardes worden bevestigd. Als tijdzone kan er natuurlijk gewoon “Europe/Amsterdam” worden gekozen, tenzij er een andere waarde gewenst is.

Om het systeem veilig te kunnen gebruiken is het nodig om een *root wachtwoord* op te geven. Dit wachtwoord is nodig om als hoofdgebruiker op het systeem in te kunnen loggen en zal dus een veilig en sterk wachtwoord moeten zijn. Onthoud en bescherm dit goed!

Na een herstart (*reboot*) kan er ingelogd worden met “root” en het gekozen wachtwoord.

3.8 GRUB installatie

De standaard bootloader van *Slackware* is *LILO*. Omdat *GRUB* gemakkelijker en uitgebreider is, gaan we in dit dictaat gebruik maken van *GRUB*.

3.8.1 Downloaden en installeren GRUB

In het geval van *GRUB* bestaat er een standaard package om te installeren, maar staat deze enkel in “extra”. Dit kunnen we vinden in de *PACKAGES.TXT*⁹. Download de *package*, en installeer deze:

```
1 | root@slackbak:~# cd /tmp/
2 | root@slackbak:/tmp# wget "ftp://ftp.fu-berlin.de/unix/linux/
   | mirrors/slackware/slackware/extra/grub/grub-0.97-i486-9.tgz"
3 | root@slackbak:/tmp# installpkg grub-0.97-i486-9.tgz
```

⁹<ftp://ftp.fu-berlin.de/unix/linux/mirrors/slackware/slackware-13.37/extra/PACKAGES.TXT>

3.8.2 grubconfig

GRUB is nu geïnstalleerd, maar nog niet geconfigureerd. Voordat *GRUB* gebruik kan worden zal dat eerst gedaan moeten worden. Dit is echter erg eenvoudig:

```
1| grubconfig |
```

Hierna krijg je de zelfde soorten vragen als bij *LILO*, je kan hier ook dezelfde antwoorden invullen.

De installatie is nu helemaal afgerond. Wanneer er gebruik wordt gemaakt van *VirtualBox* is het aan te raden een kopie te maken van de harde schijf. Wanneer er dan iets fout gaat, kan simpelweg de kopie van de schijf worden teruggezet. Dit scheelt een herinstallatie van *Slackware*.

Hoofdstuk 4

Het systeem

Nu het systeem gereed is, kan er begonnen worden met werken in de installatie. Het begint natuurlijk met een inlog prompt:

```
1| slackbak login: |
```

Na de installatie is er maar één gebruiker beschikbaar, de zogenaamde *root account*. Het wachtwoord van deze gebruiker is opgegeven tijdens de installatie. Na het inloggen kan er begonnen worden met de configuratie van het systeem.

4.1 root

De gebruiker “root” heeft alle rechten op het systeem. Het is daarom erg gevaarlijk om onder dit account te werken. Eén verkeerd commando en het systeem is vernietigd¹. Het “root” account dient daarom alleen gebruikt te worden wanneer dit noodzakelijk is. Als het dan toch gebruikt moet worden, maak dan gebruik van de volgende stappen²:

1. Ga op je handen zitten.
2. Denk na over wat je wilt doen.
3. Typ de commando's in.
4. Ga weer op je handen zitten.
5. Denk na over de consequentie van het commando.

¹rm -rf /tmp/ vs rm -rf /tmp /

²by Matt Welsh

6. Toets “enter”.
7. Log uit of ga naar 1.

Zo op het eerste gezicht zal dit misschien erg overtrokken overkomen. Iedere gebruiker die al wat langer met *UNIX* werkt zal echter kunnen vertellen dat de bovenstaande procedure een echte *lifesaver* is. De meeste mensen hebben wel een keer meegemaakt dat ze iets uitvoerden waar ze na het intoetsen van “enter” al spijt van hadden.

4.1.1 root worden

Om root te worden volstaat het om het commando `su` in te typen in de shell. Er zal nu om het wachtwoord gevraagd worden. Als dit goed is, zal de gebruiker een onbeperkte shell gepresenteerd krijgen.

4.2 Reboot, Shutdown & Runlevels

Ook een *Linux* systeem dient natuurlijk op een correcte manier gestart en gestopt te worden. Dit kan echter wel op verschillende manieren. Achter de schermen gebeurt er meestal hetzelfde, zoals het *flushen* van openstaande buffers naar de *harde schijf*, of het netjes afsluiten van draaiende *services*. Het afsluiten kan op de volgende manieren:

1. `init 0`
2. `halt`
3. `shutdown`

Een *reboot* kan op de volgende manieren:

1. `init 6`
2. `reboot`
3. `shutdown -r`

De reden dat dit op verschillende manier gedaan kan worden heeft te maken met de geschiedenis van *Linux*. Wanneer `shutdown` wordt aangeroepen, zal er eerst een *SIGTERM* signal verstuurd worden naar alle processen. Ook zal login *blocking* worden en zullen gebruikers genotificeerd worden dat het systeem down gaat. De *shutdown* of *reboot* word dus aangekondigd. Hierna

wordt er achter de schermen aan `init` gevraagd om het *runlevel* te wijzigen naar het gevraagde niveau. Wanneer er gebruik gemaakt wordt van `halt` zal er direct een *SIGKILL* signal naar de actieve processen gestuurd worden. Aangezien dit signal niet door het proces zelf kan worden genegeerd, krijgen actieve processen geen kans om gebruikers nog dingen te vragen³. Dit kan resulteren in dataverlies. Tegenwoordig zal een aanroep aan `halt` standaard worden doorgewezen naar `shutdown`, omdat dit gebruiksvriendelijker is en beter voor de consistentie van het systeem. Het zal immers een unieke gelegenheid zijn als men geen gebruik wilt maken van een nette afsluit procedure. Tegenwoordig dient er om dat doel te bereiken dan ook een speciale *flag* mee worden gegeven aan `halt`:

```
1| kevin@slackbak:~$ man halt
2| [...]
3|      -f      Force halt or reboot, don't call shutdown(8).
```

Als laatste is er `init`, welke direct de *runlevels* modificeerd. Een *runlevel* is een gevolg van een *System V* gebaseerde systeem initialisatie. Ieder *runlevel* heeft bepaalde taken die in dat niveau worden uitgevoerd. *Slackware* maakt gebruik van de volgende *runlevels*:

Runlevel	Taken
0	Halt het systeem
1	Single user mode
2,3	Multi User Mode - Default
4	3 + X11
6	Reboot het systeem

4.3 Aanmaken van gebruikers

Linux is een zogenaamd *multi-user* operating system. Dit betekent dat meerdere gebruikers gemakkelijk van hetzelfde systeem gebruik kunnen maken. Dit kan zijn via netwerk gerelateerde diensten als `remote terminals` zoals `ssh`, maar ook met meerdere hardware sets, zodat er meerdere werkplekken op een systeem ontstaan.

Voordat er van het bovenstaande gebruik gemaakt kan worden, zal een gebruiker eerst een *account* moeten krijgen op een systeem. Dit kan met behulp van het commando `adduser`.

```
1| root@slackbak:/home/kevin# adduser klaasvaak
2| Login name for new user: klaasvaak
```

³Een duidelijk voorbeeld is het afsluiten wanneer een *tekst editor* nog data heeft die nog niet is opgeslagen. Er zal nu geen dialoog worden getoond met de vraag of er nog moet worden opgeslagen

```

3 | User ID ('UID') [ defaults to next available ]:
4 | Initial group [ users ]:
5 | Additional UNIX groups:
6 | [...]
7 | :
8 | Home directory [ /home/klaasvaak ]
9 | Shell [ /bin/bash ]
10 | Expiry date (YYYY-MM-DD) []:
11 | New account will be created as follows:
12 | _____
13 | Login name.....:   klaasvaak
14 | UID.....:         [ Next available ]
15 | Initial group....: users
16 | Additional groups: [ None ]
17 | Home directory...: /home/klaasvaak
18 | Shell.....:       /bin/bash
19 | Expiry date.....:  [ Never ]

```

4.4 Terminals

Het inloggen in *Linux* gebeurt in een terminal. Standaard wordt er gestart met terminal 1. Toets “alt + F2” en terminal 2 verschijnt. Standaard start *Slackware* zes terminals. Zes verschillende of dezelfde gebruikers kunnen tegelijk ingelogd zijn. Een terminal heet *tty*:

```

1 | kevin@slackbak:~$ ps -ef | grep tty
2 | kevin 1585 1 0 20:32 tty1 00:00:00 -bash
3 | root 1586 1 0 20:32 tty2 00:00:00 /sbin/agetty 38400 tty2 linux
4 | root 1587 1 0 20:32 tty3 00:00:00 /sbin/agetty 38400 tty3 linux
5 | root 1588 1 0 20:32 tty4 00:00:00 /sbin/agetty 38400 tty4 linux
6 | root 1589 1 0 20:32 tty5 00:00:00 /sbin/agetty 38400 tty5 linux
7 | root 1590 1 0 20:32 tty6 00:00:00 /sbin/agetty 38400 tty6 linux

```

Hier is een overzicht te zien van alle *processen* die zijn gerelateerd aan de *tty*. *agetty* is een proces wat ervoor zorgt dat er een nieuwe *tty* verbinding kan worden geopend. Hierna zal er een login prompt worden gestart. Er kan dan worden ingelogd op de *terminal*. In het bovenstaande voorbeeld is te zien dat op de tweede *tty* een gebruiker met de naam “kevin” is ingelogd. Deze maakt gebruik van de *Bash* shell.

4.5 Mountpoint

Tijdens het starten van *Linux* worden er automatisch bepaalde partities beschikbaar gemaakt. Dit wordt *mounten* genoemd. Deze automatische *mounts* komen uit een speciale file, namelijk */etc/fstab*. Deze config file bevat dus de zogenaamde *statische mountpoints*. Wanneer er een bepaalde

partitie, bijvoorbeeld *root (/) gemount* moet worden, zal dit gedefiniëerd moeten zijn in *fstab*. Een voorbeeld van deze file is hieronder te zien:

```
1 kevin@slackbak:~$ cat /etc/fstab
2 /dev/sda1 / ext4 defaults 1 1
3 /dev/sda2 /home ext4 defaults 1 2
4 /dev/fd0 /mnt/floppy auto noauto,owner 0 0
5 devpts /dev/pts devpts gid=5,mode=620 0 0
6 proc /proc proc defaults 0 0
7 tmpfs /dev/shm tmpfs defaults 0 0
8 /dev/sda3 swap swap sw 0 0
```

Hier valt dus te zien dat de eerste partitie van de eerste schijf met type *EXT4* automatisch met default mount opties wordt gemount op de root van het systeem. Zo geldt het ook voor de andere regels. De mountpoints *devpts*, *proc* en *tmpfs* zijn speciale mounts.

4.5.1 Devpts

Dit mountpoint bevat alle *pseudo terminals* die geassocieerd zijn met het systeem. Alle programmatuur die gebruik maakt van input en output van userdata (*ssh* en consorten) zal van input devices moeten lezen en naar output devices moeten schrijven. Deze *pseudo terminals* zijn virtual devices die in deze behoefte voorzien. Een *ssh* sessie registreert bijvoorbeeld een virtueel toetsenbord, dat door *Bash* gebruikt kan worden om input te lezen. Om een wildgroei (en een limiet aan *pseudo devices*) te voorkomen is er een speciale *multiplexer* geschreven welke de afhandeling van alle *pseudo devices* regelt. Dit gebeurt binnen het *devpts* filesystem.

4.5.2 Proc

Dit mountpoint is een manifestatie van interne datastructuren van de *Linux* kernel. Dit is duidelijker te maken met een voorbeeld:

```
1 kevin@slackbak:~$ ps -ef | grep udevd
2 root 1016 1 0 13:49 ? 00:00:00 /sbin/udev ---daemon
3 kevin@slackbak:~$ cat /proc/1016/cmdline
4 /sbin/udev---daemon
5 kevin@slackbak:~$ cat /proc/version
6 Linux version 2.6.33.4-smp (root@midas) (gcc version 4.4.4 (GCC)
   ) #2 SMP Wed May 12 22:47:36 CDT 2010
```

Hier is dus te zien dat de *command line argumenten* van draaiende processen terug te vinden is in het *proc* file system. Veel ervan is read-only, maar sommige kernel variabelen zijn te veranderen door te schrijven naar de speciale files binnen *proc*.

4.5.3 Tmpfs

Het *mountpoint tmpfs* staat voor *temporary file system*. Sinds kernel versie 2.4 en groter is dit speciale type file system beschikbaar. Het bestaat uit gealloceerde *memory pages*. De hoeveelheid *pages* kan dynamisch groeien indien gewenst. Een andere optimalisatie is dat weinig gebruikte *pages* door de *memory manager* naar de *swap space* verplaatst kunnen worden. Ze zullen dus niet onnodig in het *RAM* blijven zitten. Dit filesystem kan worden gebruikt om data “op te slaan”, maar toch in het geheugen te kunnen bewerken. Denk bijvoorbeeld aan tijdelijke opslag.

4.5.4 Swap

Swap space is secundaire memory space voor de *Linux* kernel. Dit wordt gebruikt wanneer de hoeveelheid beschikbaar *RAM* te klein wordt. Er kan dan uitgeweken worden naar de *swap space*. De *swap* is een speciaal geformatteerde partitie, of speciale file *op* een partitie, welke gebruikt kan worden om ongebruikte *memory pages* weg te schrijven. De *swap* is wel merkbaar trager als het *RAM* geheugen. Er moet dus voorkomen worden in veel gevallen dat de *swap* gebruikt wordt.

4.5.5 Mount

Zelf een *mount* actie uitvoeren is erg simpel. Dit kan bijvoorbeeld als volgt:

```
1|mount -t vfat /dev/sdb1 /mnt/usbstick |
```

Nu wordt *device sdb*, *partitie 1* door *mount* aan */mnt/usbstick* gekoppeld. Het type van het *file system* is expliciet opgegeven als zijnde *FAT*. Dit is niet altijd nodig, aangezien moderne implementaties van *mount* zelf kunnen bepalen wat het type *file system* is. Het weglaten van de optie '-t' volstaat dan.

Met *mount* valt te zien welk *file systems* momenteel *gemount* zijn. Zoals te zien is staat de usb stick er ook bij:

```
1|kevin@slackbak:~$ mount
2|/dev/root on / type ext4 (rw,relatime,barrier=1,data=ordered)
3|proc on /proc type proc (rw)
4|sysfs on /sys type sysfs (rw)
5|usbfs on /proc/bus/usb type usbfs (rw)
6|/dev/sda2 on /home type ext4 (rw)
7|tmpfs on /dev/shm type tmpfs (rw)
8|/dev/sdb1 on /mnt/usbstick type vfat (rw) |
```

`umount` kan worden gebruikt om het *mounted file system* te ontkoppelen. Het device is dan dus niet meer beschikbaar voor het systeem:

```
1|umount /mnt/usbstick
```

Hierna kan via `mount` gecontroleerd worden of dit ook echt het geval is.

4.6 Slackware Package

Alle geïnstalleerde software is tijdens de installatie van internet gehaald. Wanneer er bepaalde pakketten vervangen moeten worden, of als er nieuwe software geïnstalleerd moet worden kan dit op twee manieren. Het gewenste pakket kan van de *Slackware* website of van de *Slackware cd-rom* gehaald worden en met de *package manager* geïnstalleerd worden. Een andere manier is de software in *source-code* vorm te installeren. Voor meer informatie over de *Slackware package manager* zie bijlage F en voor *source-code installatie* zie bijlage G.

4.7 De man pages

Eén van de belangrijkste commando's op het systeem is `man`[2][18]. Dit commando geeft een *manual* over een programma weer. Hier is in te vinden wat het programma doet en hoe het werkt. Mocht iets niet duidelijk zijn dan is met `man` altijd meer uitleg te vinden:

```
1|man man
```

Dit is alles wat er over het commando bekend moet zijn. De parameter *man* zorgt dat de *manual* van het programma `man` geopend wordt. Deze *manuals* zijn op een vaste manier opgebouwd, de structuur is bij iedere andere *manual* terug te zien.

- 1 *TITLE*: Bovenaan de pagina staat de titel van de pagina en welke *manual* wordt weergeven⁴. In dit geval is het *man(1)*.
- 2 *NAME*: Nogmaals de naam van het commando en eventuele synoniemen en een korte omschrijving.
- 3 *SYNOPSIS*: De *synopsis* of *syntax* is de manier waarop het commando gebruikt dient te worden. In het geval van `man` is dit `man [acdfFhkKtwW] [-m system] [-o string] [-C config_file] [-M path] [-P pager] [-S section_list] [section name]` Alle delen

⁴Er kunnen meerdere manual pages voor een titel zijn: `man open` en `man 3 open`

tussen de haken “[]” zijn eventuele opties. Wanneer deze worden weggelaten ontstaat er `man name`. Dit is te testen door bijvoorbeeld `man man` of `man ls`.

- 4 *DESCRIPTION*: Is een uitgebreidere omschrijving van de mogelijkheden van het commando.
- 5 *OPTIONS*: Hier worden alle mogelijke *configuratie switches* van het commando beschreven.
- 6 *SEE ALSO*: Is een overzicht van verwante commando's.

De *manual page* wordt weergegeven door middel van het programma `less`. Afsluiten kan met `q`. Voor meer info, `man less`.

Wanneer er voortaan “naar de documentatie” wordt verwezen zal, tenzij nader gespecificeerd, de *man page* van het betreffende onderwerp bedoeld worden.

4.8 Libraries

Libraries, of *bibliotheken* zijn bestanden die verschillende *functies* of *subroutines* bevatten die kunnen worden gebruikt door andere software. Op deze manier kan een systeem op een plek bepaalde *libraries* met taken aanbieden, welke dan door de rest van het systeem kunnen worden gebruikt. Niet ieder programma dat een bepaalde taak wil uitvoeren hoeft dan het wiel opnieuw uit te vinden. Het kan dan juist gebruik maken van de logica die al te vinden is in de aanwezige *libraries*.

In de loop van de tijd zijn er enorm veel *libraries* ontstaan. Als eerste komt dit omdat het ontwikkelen van applicaties veel sneller gaat. Ontwikkelaars kunnen immers code gebruiken die al bepaalde problemen oplost.

Een tweede oorzaak is de delegatie van verantwoordelijkheid. Wanneer ieder programma wat data moet *comprimeren* gebruik kan maken van dezelfde *library* zal dit allemaal dezelfde typen bestanden opleveren. Dit is erg gemakkelijk in gebruik, het ontwikkelteam van de *libraries* kan er vervolgens zorgen dat dit formaat bruikbaar blijft. Mochten er fouten gevonden worden in de implementatie, dan hoeft dit ook maar op één plek te worden opgelost, niet in ieder los programma wat wil comprimeren.

4.8.1 Static libraries

Een *static library*, of *statische bibliotheek* is een *library* die tijdens de *compilatie* van een programma wordt *gelinkt*. Dit betekent dat de benodigde

delen van de *library* worden samengevoegd met het programma.

4.8.2 Shared libraries

In tegenstelling tot *static libraries* zijn *shared libraries* niet tijdens de *compilatie* samengevoegd, maar zullen tijdens het draaien van een programma worden geopend indien nodig.

Dit gedrag heeft vele voordelen, maar de belangrijkste zijn ruimtebesparing en verminderd onderhoud.

In het geval van ruimte besparing is dit te verklaren doordat ieder programma geen eigen kopie van een bepaalde *library* hoeft te hebben. De programma's zijn netto dus kleiner.

De verbetering op het gebied van *onderhoud* komt voort uit de gevolgen van de *static libraries*. Mocht er een nieuwere versie van een *library* uitkomen omdat die veiligheidsfouten herstelt, zal dat betekenen dat ieder programma opnieuw *gelinked* moet worden aan de *library*.

Wanneer een *library* niet *static*, maar *shared* is kan er eenvoudig weg een upgrade van de betreffende *library* worden geupgrade. Hierna zal ieder programma gebruik gaan maken van de nieuwe versie, zodra dit programma herstart is. Het hele systeem is dus gelijk veilig.

ldconfig

Om te zorgen dat een systeem een idee heeft over de beschikbare *shared library* wordt er gebruik gemaakt van het configuratiebestand *ld.so.conf*, te vinden in */etc/*.

Dit bestand wordt door *ldconfig* gelezen, waarna er een cache met beschikbare *libraries* wordt gemaakt. Deze cache kan worden gebruikt door het systeem om programma's de *shared library* te geven waar ze om vragen.

Het commando *ldconfig* leest sowieso de vertrouwde *library* mappen */lib* en */usr/lib* uit, en zal vervolgens in het bestand */etc/ld.so.conf* kijken waar het nog meer naar *libraries* moet kijken.

Mocht er dus een map met *libraries* worden toegevoegd aan het systeem zal dat in dit bestand moeten gebeuren. Dit is erg eenvoudig, het bestand bevat alleen regels met locaties:

```
1| kevin@slackbak:~$ cat /etc/ld.so.conf
2| /usr/local/lib
3| /usr/i486-slackware-linux/lib
4| /usr/lib/seamonkey
```

Een nieuwe locatie is dus snel toegevoegd.

Hoofdstuk 5

Inrichting

Dit hoofdstuk gaat over de schijf indeling van een *Linux* systeem. We bedoelen niet het type *file system* waarmee de harde schijf is ingedeeld, maar juist de structuur die wordt gebruikt om data *op* de schijf te benaderen. Er zijn hier strikte richtlijnen voor opgesteld. Het volgen van deze richtlijnen wordt dan ook door veel *distributies* gevolgd, al zijn er soms kleine implementatieverschillen.

5.1 Inleiding

Om door een *file system* heen te kunnen navigeren is het handig dat de basis navigatie bekend is bij de lezer. Lees daarom minimaal de *NAME* en de *DESCRIPTION* uit de *manual* van de volgende commando's:

1. `cd` - **C**hange **D**irectory
2. `cp` - **C**opy
3. `ls` - **L**ist
4. `mkdir` - **M**ake **D**irectory
5. `pwd` - **P**rint **W**orking **D**irectory
6. `rm` - **R**emove
7. `rmdir` - **R**emove **D**irectory
8. `df` - **D**isk **F**ree

5.2 Linux Standard Base

De *Linux Standard Base*, of *LSB*[12], is een standaard die binnen de *Linux* wereld is opgestart. Men wilde hiermee een duidelijke richtlijn specificeren over de volgende onderwerpen:

1. *ELF*¹ specificatie.
2. Standaard *libraries*.
3. Standaard *commando's*.
4. *File System hiërarchie*.
5. *Lokalisatie* mogelijkheden van binaries.
6. *Systeem initialisatie*².
7. *User* en *groep* definities.
8. Print framework³.
9. *Interpreters*⁴.
10. Basis *X Window* systeem.

De meeste punten resulteren in droge opsommingen over wat er precies aanwezig moet zijn op een systeem. De informatie is duidelijk gedocumenteerd, zodat dit zelf opgezocht kan worden wanneer het nodig is. Het is echter wel handig om te weten hoe punt 4 precies in elkaar zit. Dit heeft namelijk veel praktische gevolgen voor het gebruik van een *LSB*-gecertificeerd systeem.

5.3 File System Hiërarchie

Een *Linux file systeem* is te representeren als een grote boom. De root node van deze tree is /, welke 0-n⁵ verschillende *child nodes*⁶ bevat. Deze *child nodes* kunnen zelf ook weer *child nodes* hebben. Dit is te visualiseren in figuur 5.1. Hier is een bepaalde *node kevin* te zien, welke dus een diepte niveau van 3 heeft.

¹*Executable and Linking Format - Binary interface standaard*

²*Run levels, init scripts, ...*

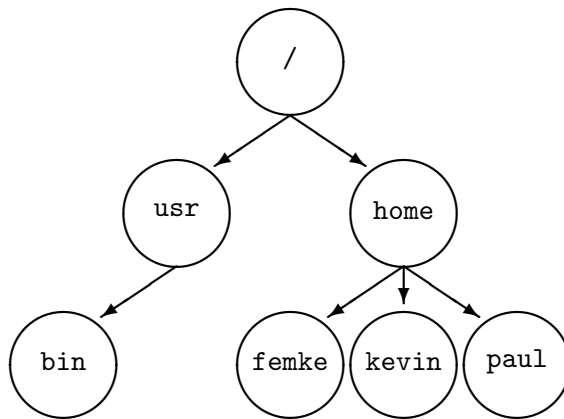
³*CUPS*

⁴*python, perl, ...*

⁵n is afhankelijk van het filesystem. *EXT3* heeft bijvoorbeeld een limiet van 31998.

⁶Directe kinderen, dus bestanden of mappen *in* een *parent*, een map.

1. /
2. /home
3. /home/kevin



Figuur 5.1: Een file system tree

Alle *nodes* op niveau 2 hebben echter een speciale betekenis. Ze bevatten allemaal *content* die aan hun beschrijving voldoet. Dit maakt het gemakkelijk en voorspelbaar om bepaalde kritieke bestanden te vinden, te gebruiken of aan te passen. Een goed voorbeeld om duidelijk te maken wat er precies bedoeld wordt zijn configuratie bestanden. Deze zijn in principe altijd in */etc* terug te vinden. De volgende *directories* zijn beschreven met hierbij hun doel:

Directory	Beschrijving
/bin	Basis user-gerelateerde programma's en scripts, zoals <i>ls</i> .
/boot	<i>Bootloader</i> (<i>LILO</i> , <i>GRUB</i>) files.
/dev	<i>Block-</i> , en <i>Character devices</i> .
/etc	Configuratie en systeem initialisatie.
/home	Gebruikersmappen (geen <i>root</i>).
/lib	Belangrijke libraries (<i>kernel modules</i> , <i>libc</i>).
/mnt	Directory voor generieke <i>mountpoints</i> .
/opt	Optionele packages. <i>Slackware</i> installeert <i>KDE</i> hier.
/proc	<i>Proc</i> filesystem voor <i>kernel</i> datastructuren.
/root	<i>Root's</i> home directory.
/sbin	Systeem programma's, nodig voor startup of <i>root</i> .
/tmp	Tijdelijke bestanden, iedereen heeft lees + schrijf rechten hier.
/usr	Gebruikers-gerelateerde programma's (browser, <i>X11</i> , ...).
/var	Systeem log files, <i>lock files</i> , <i>mail spools</i> en <i>printer spools</i> .

Als tradities had de “root” user zijn home directory oorspronkelijk in /, maar veel distributies gebruiken tegenwoordig */root/*[23]. Dit heeft dus oorspronkelijk niets te maken met de beveiliging van het systeem.

Hoofdstuk 6

Editors

Op een *Linux* systeem zijn er verschillende *editors*, ofwel tekstverwerkers beschikbaar. De verschillende *editors* hebben allemaal een eigen doelgroep. Zo zijn er *editors* die zich richten op beginnende gebruikers, zoals `pico`, maar zijn er ook erg geavanceerde programma's als `emacs` of `vim`.

Een programma als `pico` is gemakkelijk in het gebruik, er is weinig voorkennis vereist om ermee te werken. Bij *editors* als `emacs` of `vim` is dit het tegenovergestelde. Dit betekent echter wel dat, eenmaal hieraan gewend, dat deze ontzettend snel werken. Verder hebben ze als voordeel dat ze gebruikers beschermen voor dingen als *RSI*, omdat muis gebruik wordt vermeden.

Vanuit historisch oogpunt is er altijd een laconieke vete geweest tussen twee kampen; `emacs` en `vim`. Dit wordt de *editor wars* genoemd. Er is geen enkele manier om sneller een discussie aan te wakkeren dan te melden dat een van de twee “veel beter” is dan de ander.

6.1 Pico

`pico` is een simpel scherm georiënteerde tekst editor. Om de *teksteditor* `pico` te starten volstaat het om het volgende uit te voeren:

```
1| kevin@slackbak:~$ pico document.txt
```

Het bestand *document.txt* zal nu worden geopend. Mocht het bestand niet bestaan dan zal `pico` nog steeds opstarten en beginnen in een nieuw bestand met de naam *document.txt*.

`nano` is een uitgebreide/verbetererde versie van `pico`. Bij veel distributies zal standaard zowel `pico` als `nano` geïnstalleerd zijn. De interface en toetsencombinaties in `nano` werken hetzelfde als die in `pico`.

6.1.1 Bediening

Wanneer `pico` start zal er onderin het scherm het een en ander aan mogelijkheden staan. Deze opties zijn te gebruiken met de toetscombinatie die ervoor staat. Het dakje `^` staat voor `ctrl`. De toetsencombinatie `^X` betekent dus `ctrl-X`. De letters zijn niet hoofdlettergevoelig.

Een ander handigheidje binnen `pico` is het gebruik van de *kill buffer*. Wanneer de combinatie `^K` wordt gebruikt, zal de huidige regel worden geknipt. Als hij weer moet worden geplakt kan dit gedaan worden met `^K`.

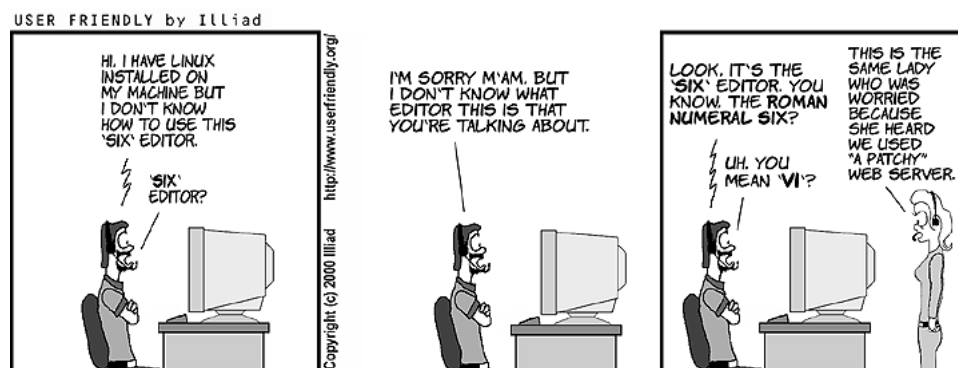
Zodra er in de huidige *buffer* een wijziging heeft plaatsgevonden zal de tekst *modified* onder in beeld komen. Wanneer `pico` nu afgesloten wordt zonder het bestand op te slaan, zal deze uit zichzelf een waarschuwing laten zien. Er kan dan alsnog worden gekozen om het bestand op te slaan.

6.2 VI(M)

The number of the beast: VI VI VI

De bekendste `UNIX` editor is zonder twijfel `vi`. Deze editor is op elk `UNIX` systeem standaard geïnstalleerd en door velen gehaat. `vim` is een verbeterde versie van `vi` en werkt vele malen gemakkelijker.

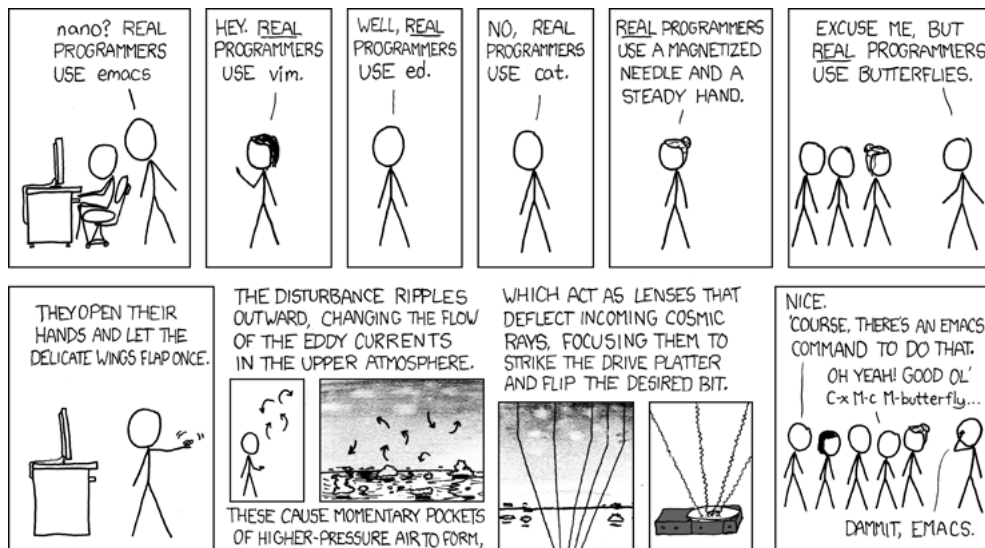
`vim` is in hoge mate configureerbaar en heeft vrijwel eindeloze mogelijkheden. De bijlage J beschrijft enkele configuratie mogelijkheden van `vim`.



Figuur 6.1: Van:userfriend.org

6.3 Emacs

Omdat er ook mensen waren die met een *handige editor* wilden werken, is er ook één ontstaan met de naam **emacs**. Deze *editor* is nog modulairder dan **vim** en is volgens velen ook handiger in gebruik, al is dit een gegarandeerd discussiepunt. In bijlage I is een kleine inleiding tot het gebruik van **emacs** te vinden.



Figuur 6.2: Van:xkcd.org

Hoofdstuk 7

User management

Linux biedt uitgebreide mogelijkheden tot *user management*. Omdat de dingen die in dit hoofdstuk worden uitgelegd voor meer *Unices* gelden, zal het kennis zijn die altijd van pas zal komen. Wanneer de uitleg als kort wordt ervaren kan er door de lezer altijd worden teruggegrepen op de `man` pages van het betreffende onderwerp.

7.1 Commando's

Het is natuurlijk erg handig om de mogelijkheid te hebben om te kijken wie er momenteel op een systeem bezig is, of wie er ingelogd is geweest. Het zien, en eventueel veranderen van gebruikersinformatie behoort ook tot de mogelijkheden.

7.1.1 `who`

`who` is een commando dat laat zien welke gebruikers er momenteel zijn ingelogd. Ook is te zien welke terminal ze gebruiken en wanneer ze zijn ingelogd:

```
1| kevin@slackbak: ~ $ who
2| kevin pts/0 2010-12-08 08:47 (145.24.213.121)
3| paul pts/1 2010-12-08 09:09 (145.24.213.56)
```

7.1.2 `w`

Het commando `w` laat zien wie er is ingelogd en welke taak ze momenteel uitvoeren. Ook wordt de huidige *load* van het systeem geprint:

```

1| kevin@slackbak:~$ w
2| 09:10:13 up 19:21,  2 users,  load average: 0.00, 0.00, 0.00
3| USER  TTY  FROM          LOGIN@  IDLE   JCPU   PCPU   WHAT
4| kevin  pts/0 145.24.213.121 08:47   0.00s  0.02s  0.00s  w
5| paul   pts/1 145.24.213.56  09:09   4.00s  0.04s  0.03s  emacs

```

7.1.3 last

`last` kan gebruikt worden om te zien welke gebruikers er wanneer hebben ingelogd in het systeem:

```

1| kevin@slackbak:~$ last
2| paul   pts/1 145.24.213.56  Thu Dec 9 13:01 - 17:42 (04:40)
3| kevin  pts/0 145.24.213.121 Thu Dec 9 12:52 - 13:06 (00:13)
4| kevin  pts/0 145.24.213.121 Thu Dec 9 12:49 - 12:51 (00:02)

```

7.1.4 getent

Het commando `getent` wordt gebruikt om *entries* uit administratieve databases te laten zien. Er kan dus gebruikersinformatie opgehaald worden. Welke databases er worden gebruikt hangt af van de configuratie van het systeem. Standaard zullen dit waarschijnlijk de bestanden `/etc/passwd` en `/etc/group` zijn voor gebruikers en groeps informatie.

Om de informatie van een gebruiker met de naam “kevin” te laten zien kan er het volgende gedaan worden:

```

1| kevin@slackbak:~$ getent passwd kevin
2| kevin:x:1000:100:Kevin van der Vlist , , ,:/home/kevin:/bin/bash

```

7.1.5 usermod

`usermod` stelt een beheerder in staat om gebruikersinformatie aan te passen. Het onderstaande commando laat zien hoe de gebruiker “kevin” wordt toegevoegd aan de groep “wheel”:

```

1| usermod -a -G wheel kevin

```

7.1.6 chown

Het commando `chown` stelt een gebruiker in staat om de eigenaar van een bestand te veranderen. Dit kan natuurlijk alleen als de gebruiker voldoende rechten heeft voor de beoogde wijziging. Het is als “root” gebruiker altijd mogelijk om dit commando uit te voeren.

```

1| kevin@slackbak:~$ getent group projectgroep
2| projectgroep:x:1001:kevin , paul
3| kevin@slackbak:~$ mkdir map
4| kevin@slackbak:~$ chmod 750 map
5| kevin@slackbak:~$ ls -l
6| total 4
7| drwxrwx— 2 kevin users 4096 Dec  8 10:45 map
8| kevin@slackbak:~$ chown kevin:projectgroep map/
9| kevin@slackbak:~$ ls -l
10| total 4
11| drwxrwx— 2 kevin projectgroep 4096 2010-12-08 10:45 map/

```

Hier is te zien dat er een nieuwe map wordt aangemaakt waar de gebruikers “paul” en “kevin” in kunnen werken. Op deze manier kan er dus op een veilige manier informatie gedeeld worden. Een gebruiker buiten *projectgroep* kan immers niet de map in.

7.2 User Identifiers

Een *user identifier*, of *uid*, is een uniek getal dat een gebruiker op een *Linux* systeem toegewezen krijgt. Dit getal wordt gebruikt om een gebruiker te identificeren, bijvoorbeeld als eigenaar van een bestand. Er kunnen dus geen twee gebruikers met hetzelfde *uid* voorkomen.

De “root” gebruiker is een speciale gebruiker. Deze krijgt het *uid* 0. Wanneer er in de *source code* van de *Linux kernel* wordt gekeken zal er ook zichtbaar zijn dat er bij belangrijke operaties wordt gekeken of het *uid* van de actieve user 0 is. Er wordt hier wel onderscheid gemaakt tussen een *real user id* en een *effective user id*.

7.2.1 Real User ID

Het *real uid* is het echte *uid* van een gebruiker. Dit is het nummer wat te zien is wanneer de gebruikers informatie wordt opgevraagd. Dit is dus het nummer dat ook wordt gebruikt om bestanden aan de gebruiker te koppelen.

7.2.2 Effective User ID

Het *effective uid* is nagenoeg altijd hetzelfde als het *real uid*. Wanneer het *sticky bit* is ingesteld op een bepaald commando zal er pas iets veranderen. De gebruiker behoudt dan zijn *real uid*, maar verandert zijn *effective uid* naar het uid van de *eigenaar* van het bestand. Laten we dit illustreren met een voorbeeld:

```

1| kevin@slackbak:~$ ls -ln /usr/bin/passwd
2|-rws-x-x 1 0 0 56255 Feb 28 2010 /usr/bin/passwd

```

Hier is te zien dat het `passwd` commando van de gebruiker met uid 0 is, dus van “root”. We zien echter ook dat het *set-uid* bit is aangezet¹. Wanneer we dit als gebruiker uitvoeren zal tijdens het commando het volgende gelden:

Wat	Waarde
Real UID	1000
Effective UID	0

Dit zorgt ervoor dat het programma toegang krijgt tot bestanden die eigenlijk alleen voor de “root” gebruiker beschikbaar zijn.

7.3 File mode bits

File mode bits worden gebruikt om bij te houden of gebruikers rechten hebben om een bepaalde file of map te openen of te beschrijven. Deze rechten worden per bestand of map opgeslagen. Het manifesteert zich als volgt:

```

1| kevin@slackbak:~$ ls -l /home/
2| total 28
3| drwxr-xr-x 2 root root 4096 Feb 13 2010 ftp
4| drwxr-x-x 3 kevin users 4096 Dec 8 09:39 kevin
5| drwx----- 2 root root 16384 Dec 6 13:25 lost+found
6| drwx-x-x 2 paul users 4096 Dec 8 09:10 paul
7| -rw-r-r- 1 kevin users 0 Dec 8 09:58 bestand

```

De eerste kolom is de belangrijkste in dit geval. Als voorbeeld wordt er naar de map *kevin* gekeken:

```
1| drwxr-x-x
```

De eerste letter, de *d*, kan verschillende waardes hebben. Elke waarde staat voor een aparte categorie. Deze zijn als volgt:

Wat	Betekenis
-	file
d	directory
b	block special file
c	character special file
l	symbolic link
p	named pipe
s	domain socket

Hierna word er naar groepen van 3 karakters gekeken. De eerste kolom

¹Dit is aangegeven met de *s* in de `chmod` waarde, zie `man chmod` voor meer informatie.

staat voor de rechten van de eigenaar, de tweede groep staat voor de rechten van de groep en de derde staat voor de rechten voor world, de rest. De *r* staat voor read, *w* staat voor write en *x* betekend execute. Wanneer we de bovenstaande rechten erbij pakken, zien we het volgende:

Wie	Rechten	Betekenis
owner	rwX	Lezen, schrijven en uitvoeren.
group	r-x	Lezen en uitvoeren.
other	x	Uitvoeren.

Aangezien het hier een map betreft heeft *uitvoeren* een speciale betekenis, namelijk de rechten om een map te openen. Het is belangrijk om hier rekening mee te houden:

```

1| kevin@slackbak:~$ mkdir map
2| kevin@slackbak:~$ touch map/file
3| kevin@slackbak:~$ ls -lhR map/
4| map/:
5| total 0
6| -rw-r--r-- 1 kevin users 0 Dec  8 10:30 file
7| kevin@slackbak:~$ chmod 600 map/
8| kevin@slackbak:~$ ls -lhR map/
9| map/:
10| ls: cannot access map/file: Permission denied
11| total 0
12| -????????? ? ? ? ?           ? file

```

Zoals hierboven te zien is, kan het “gekke” resultaten opleveren wanneer een bestand genoeg rechten heeft, maar de map waar deze in staat niet.

Voor normale bestanden betekend de *uitvoeren* optie dat het bestand uitgevoerd mag worden als shell script. Wanneer dit niet ingesteld is, zal je een *permission denied* foutmelding krijgen op het bestand:

```

1| paul@slackbak:~$ ls -l
2| total 0
3| -rw-r--r-- 1 paul users 0 2010-12-15 23:37 test.sh
4| paul@slackbak:~$ ./test.sh
5| -bash: ./test.sh: Permission denied

```

Ook als er gebruik gemaakt wordt van een taal zoals bijvoorbeeld *perl*, *python* of *php* moet je de permissies op *execute* zetten het bestand te draaien.

7.3.1 Waardes

De volgende waardes worden gebruikt om rechten aan te geven:

waarde	binair	betekenis
1	001	Uitvoeren
2	010	Schrijven
4	100	Lezen

Deze waardes worden in de praktijk gebruikt voor `chmod`

```
1| kevin@slackbak:~$ chmod 600 map
```

Dit commando stelt voor de categorie *user* de rechten op 6, binair *110*. Dit betekent dus lezen, schrijven en niet uitvoeren. De rest staat op 0, dus geen toegang.

7.3.2 Gebruik van numerieke waarden

In veel voorbeelden, zowel in dit document als buiten dit document, zal er gebruikt gemaakt worden van de numerieke varianten, in plaats van de letters. Het is dus belangrijk dat je weet wat de verschillende numerieke waarden betekenen.

7.3.3 umask

Om te kunnen beïnvloeden onder welke rechten een bestand wordt aangemaakt door een proces kan er gebruik gemaakt worden van een `umask`. Dit staat voor **user mask**. Het betekent dat een proces bepaalde bits gekoppeld aan de file mode bits weghaald. Dit wordt met een bitwise operatie gedaan, bijvoorbeeld $777 \& (\sim 022) = 755$. Dit is gemakkelijker te zien met een klein voorbeeld:

```
1| kevin@slackbak /tmp/demo$ umask 755
2| kevin@slackbak /tmp/demo$ touch file
3| kevin@slackbak /tmp/demo$ umask 177
4| kevin@slackbak /tmp/demo$ touch prive
5| kevin@slackbak /tmp/demo$ umask 133
6| kevin@slackbak /tmp/demo$ touch hoi
7| kevin@slackbak /tmp/demo$ umask 100
8| kevin@slackbak /tmp/demo$ touch dag
9| kevin@slackbak /tmp/demo$ ls -lh
10| total 0
11| -rw-rw-rw- 1 kevin users 0 2011-04-13 10:24 dag
12| -----w-w- 1 kevin users 0 2011-04-13 10:24 file
13| -rw-r--r-- 1 kevin users 0 2011-04-13 10:24 hoi
14| -rw----- 1 kevin users 0 2011-04-13 10:24 prive
```

Hier is te zien dat een `umask` van 755 resulteert in het weghalen van alle rechten, behalve *read* (octaal 2). Wanneer het `umask` vervolgens wordt veranderd naar iets heel anders, bijvoorbeeld 100, blijkt dat er lees en schrijf rechten overblijven.

Hoofdstuk 8

Proces management

Een *proces* is een actief programma op een machine. Onder actief wordt verstaan dat het programma een gedeelte van de *stysteem resources* wil gebruiken. Hiervoor zal het echter wel van de *kernel* toegang moeten krijgen.

8.1 Processen bekijken

Er zijn verschillende manieren om te zien welke *processen* er actief zijn. De meest gebruikte varianten zijn `ps` en `top`.

```
1 kevin@slackbak:~$ man ps
2 [...]
3 -e          Select all processes. Identical to -A.
4 -f          does full-format listing. [...]
5 kevin@slackbak:~$ ps -ef
6 UID      PID PPID  C STIME TTY   TIME     CMD
7 root     1    0    0 Dec07 ?     00:00:02 init [3]
8 root     2    0    0 Dec07 ?     00:00:00 [kthreadd]
9 [...]
10 root    2227 1406 0 10:48 ?     00:00:00 sshd: kevin [priv]
11 kevin   2229 2227 0 10:48 ?     00:00:00 sshd: kevin@pts/0
12 kevin   2230 2229 0 10:48 pts/0 00:00:00 -bash
13 kevin   2280 2230 0 11:08 pts/0 00:00:00 ps -ef
```

Wanneer deze listing begrepen wordt is er veel informatie uit te halen. Aan gezien dit belangrijke en handige informatie betreft, zullen alle kolommen de nodige uitleg krijgen.

8.1.1 UID

Deze kolom bevat het *effective uid* van het proces op de betreffende regel. Hier blijkt dus onder welke gebruiker een *proces* effectief draait.

8.1.2 PID

In deze kolom is het *proces id*, of *pid* te zien. Dit *proces id* is een unieke identifier van een *proces*, en kan daarom slechts eenmalig voorkomen. Dit *pid* wordt gebruikt wanneer een bepaald proces moet worden gespecificeerd.

8.1.3 PPID

Dit is het *parent proces id* van een *proces*. Dit geeft het *pid* weer van het proces wat (meestal met behulp van *fork()*) een *child* proces heeft gestart. De relatie van een *proces* met zijn parent is een heel belangrijke. Een proces kan namelijk niet leven zonder *parent*. Als de parent sterft, zal het child proces ook mee gaan, tenzij deze is *detached*.

8.1.4 C

Dit is de integer representatie van het cpu gebruik. Het cijfer wordt als volgt berekend:

$$\frac{cputijd}{echte_tijd} \cdot 100 = (int)bezetting$$

cputijd is hierbij de tijd waarin een proces zich in *running state* bevindt, *echte_tijd* is de echte tijd. Het proces om deze *PDF* te maken heeft de volgende kenmerken:

1	real	0m15.877 s	
2	user	0m8.221 s	

Cputijd zou hier dus zijn:

$$\frac{8.221}{15.877} \cdot 100 = 51.7793034$$

$$(int)bezetting = 51$$

8.1.5 STIME

De kolom *STIME* bevat de start time van het *proces*.

8.1.6 TTY

In de kolom *TTY* staat de *terminal* waar het proces draait.

8.1.7 TIME

Dit is de hoeveelheid *cputijd* die een proces heeft gebruikt. Hoe hoger *TIME* is, hoe actiever een *proces* dus is.

8.1.8 CMD

In deze laatste kolom is de naam van het *proces* te zien, samen met de betreffende parameters.

8.2 Termination

Wanneer een *proces* niet meer reageert, of taken gaat uitvoeren die niet binnen zijn takenpakket vallen, kan het zijn dat een proces afgeschoten moet worden. Het commando `kill` biedt deze mogelijkheid, door een *signal* naar het proces te sturen. Deze *signals* zijn onderdeel van de *POSIX* standaard. De twee belangrijkste signals voor nu zijn *SIGTERM* en *SIGKILL*. Om te kijken wat dit precies betekend zou je in `man 7 signal` kunnen kijken.

Het eerste *signal* geeft een proces te kennen dat het beëindigd moet worden. Het proces zal dan afsluiten:

```
1| kevin@slackbak:~$ sleep 10&
2| [1] 2427
3| kevin@slackbak:~$ kill 2427
4| kevin@slackbak:~$ ps -ef | grep sleep
5| kevin      2429  2230  0 12:28 pts/0    00:00:00 grep sleep
6| [1]+  Terminated          sleep 10
```

Een *proces* kan een *SIGTERM* signal afvangen en negeren. Een andere mogelijkheid is dat een *proces* ongedefinieerd gedrag gaat vertonen. Beide situaties kunnen resulteren in een proces dat niet is te stoppen door een *SIGTERM* te sturen:

```
1| kevin@slackbak:~$ top&
2| [1] 2430
3| kevin@slackbak:~$ kill 2430
4| [1]+  Stopped              top
5| kevin@slackbak:~$ ps -ef | grep top
6| kevin      2430  2230  0 12:28 pts/0    00:00:00 top
7| kevin      2434  2230  0 12:29 pts/0    00:00:00 grep top
```

```

8| kevin@slackbak:~$ kill -9 2430
9| kevin@slackbak:~$ ps -ef | grep top
10| kevin      2436   2230   0 12:29 pts/0    00:00:00 grep top
11| [1]+  Killed                  top

```

Het *SIGHUP* signal kan ook door programmeurs worden afgevangen. Dit signal wordt veel gebruikt om een *proces* te laten herstarten. De werking is weer hetzelfde.

```
1| kevin@slackbak:~$ kill -HUP 1337
```

Na het uitvoeren van het `kill` commando met *PID* 1337 als parameter, zal het *proces* met 1337 als *PID* herstarten zonder dat deze afsluit.

8.3 Zombies

Zombie processen zijn een speciaal soort processen. Deze processen zijn klaar met het uitvoeren van hun taken en zouden dus moeten verdwijnen uit de *process table*. De *parent* van een *proces* heeft echter nog de mogelijkheid om de exit status van een *child proces* te bekijken. Wanneer dit niet gebeurt, zal het proces nog aanwezig blijven in de *process table*. *Zombie's* zijn per definitie *processen* die geen *resources* gebruiken, dus de enige 'last' die een gebruiker ervan ondervindt is de entry in de *process table*.

Een *zombie proces* afschieten is niet altijd mogelijk. Wanneer een *zombie proces* een *parent pid* heeft, is het mogelijk om de *parent* af te sluiten. *Childs* zullen hierdoor ook sterven, waardoor de *zombie* weg is.

Wanneer een *zombie init* (pid 1) als *parent* heeft, zal het onmogelijk zijn het *proces* af te sluiten. De enige manier om de *zombie* dan weg te krijgen is een complete reboot.

8.4 Load

Load[25] is een belangrijk getal in de *Linux* wereld. De getallen bij *load* geven aan hoe druk de *CPU* is, maar het interpreteren van het getal gebeurt vaak fout.

De *load* kan via diverse commando's opgevraagd worden, te weten:

- `uptime`
- `w`
- `top`

- De file `/proc/loadavg`

Alle commando's zullen (In principe) dezelfde waarde teruggeven als resultaat.

Bij een computer die niets zal te doen zal de *load* 0.00 zijn.¹ Wanneer je het commando `load` uitvoert komen er drie cijfers uit:

```
1| paul@slackbak:~$ uptime
2| 16:47:37 up 6 days, 17:37, 1 user, load average: 0.00, 0.00,
   | 0.00
```

Zoals te zien heeft onze server momenteel niets te doen.

```
1| srv01:~# uptime
2| 17:05:30 up 224 days, 3:01, 1 user, load average: 2.37,
   | 3.42, 0.39
```

Deze server heeft het een stuk drukker. De drie waardes geven de *load* aan van de laatste minuut, vijf minuten en vijftien minuten. In het geval van hierboven staat er:²

- In de laatste minuut was de *CPU* overbelast met *137%*.
- In de laatste vijf minuten was de *CPU* overbelast met *242%*.
- In de laatste vijftien minuten was de *CPU* onderbelast met *61%*.

Wanneer de *CPU* 1,37 keer sneller was geweest als de huidige *CPU* dan had de *CPU* alle taken in die minuut kunnen uitvoeren. Een *load* van precies 1.00 betekend dat de *CPU*, indien er één core is, volledig gebruikt wordt voor die minuut. Als je bijvoorbeeld 4 cores ter beschikking hebt en een *load* van 4.00, betekend dit hetzelfde als een *load* van 1.00 bij een enkele core.

¹ *UNIX* en *Linux* verschillen in het berekenen van de *load*. *UNIX* gebruikt alleen de processen welke in *RUNNING* of *RUNNABLE* state zijn, terwijl *Linux* ook de processen in de *uninterruptible* state meerekent.

²Let op: Dit voorbeeld gaat uit van één *CPU core*. Wanneer er meer als één core is, dan moet er in plaats van één het aantal cores van de *load* afgehaald worden.

Hoofdstuk 9

Linux networking

Linux is enorm geschikt om te gebruiken in netwerk gerelateerde toepassingen. Doordat het een *multi-user* systeem is, is het erg gemakkelijk om gebruikers via het netwerk diensten aan te bieden. Voordat een systeem bereikbaar is moet er natuurlijk wel een netwerk configuratie gedaan worden. Daarna willen we een paar veel gebruikte services aanduiden.

9.1 Configuratie

Het netwerk kan op twee manieren worden ingesteld. Er kan gekozen worden om een automatische configuratie te doen, maar er kan ook voor de handmatige manier gekozen worden. We zullen beide manieren beschrijven.

9.1.1 netconfig

Om het netwerk in te stellen is er een script wat je hierbij helpt. Dit script heet **netconfig**. De werking hiervan is enorm eenvouding. Er worden per stap wat vragen aan je gesteld. Geef antwoord op de vragen, en bevestig daarna je instellingen. **netconfig** zal ze dan toepassen.

```
1| root@slackbak:~/home/kevin# netconfig
```

9.1.2 Handmatige configuratie

Om het netwerk handmatig te configureren is het nodig om de netwerk config file aan te passen. Dit bestand bevat alle netwerk gerelateerde configuratie. We zullen hier de relevante informatie geven van een bedraad device onder de naam *eth0*.

```

1 root@slackbak:/home/kevin# cat /etc/rc.d/rc.inet1.conf
2 [...]
3 # If USE_DHCP[interface] is set to "yes"
4 # this overrides any other settings.
5 IPADDR[0]="145.24.222.162"
6 NETMASK[0]="255.255.255.0"
7 USE_DHCP[0]=""
8 DHCP_HOSTNAME[0]=""
9 [...]
10 # Default gateway IP address:
11 GATEWAY="145.24.222.253"
12 [...]
13 kevin@slackbak:~$ cat /etc/HOSTNAME
14 slackbak.cmi-hro.nl

```

Zoals je kunt zien is er niets wereld schokkends aan de config file. Alle velden spreken voorzich. Wanneer het systeem up komt, zullen bovenstaande waardes uitgelezen worden waarna het systeem ze zal gebruiken.

9.1.3 /etc/hosts

Deze file bevat verwijzingen naar hosts op het netwerk, welke niet door de *DNS* server gegeven zullen worden opgezocht. De syntax vereist geen uitleg:

```

1 127.0.0.1          localhost
2 145.24.222.162    slackbak.cmi-hro.nl slackbak
3 145.24.129.221    ftphro.nl ftphro

```

9.1.4 /etc/resolv.conf

Om de *DNS* in te stellen kan de file */etc/resolv.conf* gebruikt worden. *DNS* servers toevoegen gaat erg gemakkelijk:

```

1 root@slackbak:/home/kevin# cat /etc/resolv.conf
2 nameserver 145.24.129.1
3 nameserver 145.24.129.2

```

9.1.5 ifconfig

Om netwerk instellingen te doen zonder config files aan te passen en services te reloaden is er ook een tool genaamd *ifconfig* beschikbaar. De output hiervan is behoorlijk intimiderend. Wanneer er echter rustig naar gekeken word, vallen veel regels al op zijn plaats. Wanneer de velden niet duidelijk zijn verwijzen we weer door naar de *man pages*.

```

1 kevin@slackbak:~$ /sbin/ifconfig

```

```

2 eth0 Link encap:Ethernet HWaddr 00:0c:29:fa:16:57
3   inet addr:145.24.222.162 Bcast:145.24.222.255 Mask
   :255.255.255.0
4   inet6 addr: fe80::20c:29ff:fefa:1657/64 Scope:Link
5   UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
6   RX packets:16340 errors:0 dropped:0 overruns:0 frame:0
7   TX packets:5796 errors:0 dropped:0 overruns:0 carrier:0
8   collisions:0 txqueuelen:1000
9   RX bytes:1858849 (1.7 MiB) TX bytes:1138260 (1.0 MiB)
10  Interrupt:18 Base address:0x1400
11
12 lo   Link encap:Local Loopback
13   inet addr:127.0.0.1 Mask:255.0.0.0
14   inet6 addr: ::1/128 Scope:Host
15   UP LOOPBACK RUNNING MTU:16436 Metric:1
16   RX packets:0 errors:0 dropped:0 overruns:0 frame:0
17   TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
18   collisions:0 txqueuelen:0
19   RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

```

Deze uitvoer laat de instellingen van twee netwerk devices zien. Namelijk *eth0* en *lo*. Dit zijn respectievelijk de netwerk-kaart en het loopback device. De netwerkkaart (*nic*) is ingesteld met het commando `netconfig` en het loopback device (*lo*) is altijd aanwezig, ook al is er geen *nic* aanwezig is.

```
1|root@slackbak:/home/kevin# ifconfig eth0 down
```

Met het commando `ifconfig` kunnen de network devices ingesteld en verwijderd worden. De manual page van `ifconfig` zegt dat de synopsis van `ifconfig`: 'ifconfig interface options' is. Het commando `ifconfig eth0 down` zorgt ervoor dat de device *eth0* verwijderd wordt uit de network configuratie.

9.2 SSH

9.2.1 Het verleden

Om via het netwerk op een andere machine te werken kan er gebruik gemaakt worden van een remote terminal. De software die een remote terminal mogelijk maakt is in feite niets meer dan een virtueel toetsenbord en een virtuele output device. In het verleden werd hier *telnet* voor gebruikt.

Een van de grootste nadelen van *telnet* is het gemak waar de communicatie mee afgeluisterd kan worden. Telnet verstuurt al zijn verkeer namelijk onversleuteld. Wanneer je via telnet afgeluisterd zou worden, word dus ook het "root" wachtwoord van de betreffende server gecompromitteerd. Omdat servers vaak grote hoeveelheden gevoelige data bevatten is dit een groot

gevaar voor de integriteit van onze moderne digitale infrastructuur.

9.2.2 OpenSSH

Om toch veilig te kunnen werken op machines waar je via het netwerk mee verbonden bent is er een programma ontwikkeld wat bekend staat onder de naam *openssh*. Deze naam staat voor 'open secure shell', een techniek die van oorsprong uit de *BSD* wereld komt. Deze techniek is dankzij het open karakter van de *BSD-style* licenties uitgegroeid tot een industrie standaard voor remote terminals.

OpenSSH bestaat uit twee delen, een server en een client. De server luistert standaard op poort 22 naar inkomende verbindingen geïnitieerd door de OpenSSH client. Omdat SSH een fundamenteel onderdeel is voor onderlinge communicatie tussen servers, maar ook voor onderhoud zullen we in appendix H een overzicht geven over de werking van SSH.

9.2.3 De eerste keer

Wanneer er voor het eerst verbinding wordt gemaakt met een host zal het volgende te zien zijn.

```
1| kevin@slackbak:~$ scp kevin@host.nl:/home/kevin/backup.tar.gz ~/
   | backup.gz
2| The authenticity of host 'host.nl (1.2.3.4)' can't be
   | established.
3| RSA key fingerprint is 0e:35:8e:be:6a:61:f0:dd:b1:ef:fe:a6:7f:f6
   | :a5:da.
4| Are you sure you want to continue connecting (yes/no)? yes
5| Warning: Permanently added 'host.nl,1.2.3.4' (RSA) to the list
   | of known hosts.
6| kevin@host.nl's password:
```

Hiermee wordt de identiteit van een host gecontroleerd. In de toekomst zal dit gebruikt worden om te verifiëren of er weer met dezelfde host verbinding wordt gemaakt.

Het is dus belangrijk om te zorgen dat de *RSA key fingerprint* correct is. Dit is te controleren door deze handmatig te vergelijken met de fingerprint van de server. Vraag bij de beheerder de key op, of controleer hem zelf na de configuratie van server:

```
1| kevin@slackbak:~$ ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key
2| 2048 0e:35:8e:be:6a:61:f0:dd:b1:ef:fe:a6:7f:f6:a5:da /etc/ssh/
   | ssh_host_rsa_key.pub (RSA)
```


9.4 FTP

Een protocol wat van traditioneel veel gebruikt word om binaire data over te sturen is *FTP*, het *file transfer protocol*. Dit protocol is te gebruiken met het ftp commando. De meest simpele werking is als volgt.

```
1| kevin@slackbak:~$ ftp ftp.hro.nl
2| Connected to orpheus.hro.nl.
3| [...]
4| 220 Service Ready for new User
5| Name (ftp.hro.nl:kevin):
```

of

```
1| kevin@slackbak:~$ ftp
2| ftp> open ftp.hro.nl
3| Connected to orpheus.hro.nl.
4| [...]
5| 220 Service Ready for new User
6| Name (ftp.hro.nl:kevin):
```

Op de regel met *Name* staat 'ftp.hro.nl:kevin'. Dit betekend dat, wanneer je niets invult, er met de user kevin ingelogd zal worden. Dit word gedaan omdat dit de username op je huidige systeem is. Gebruik als username *studentno.extensie* en als password je HRO wachtwoord. Je bent nu ingelogd op de FTP-server van de HRO.

Onthoud wel dat *ftp* de data onversleuteld verstuurd. Wanneer je een verbinding dus niet vertrouwd is het af te raden om gebruik te maken van FTP.

9.5 SFTP

Omdat *ftp*, zoals vermeld, gebruikt maakt van onversleutelde kanalen kan het reden zijn dit niet te gebruiken. Niemand wil natuurlijk dat zijn account gegevens van een machine op straat komen te liggen. Er kan om deze reden dan ook gebruik gemaakt worden van *sftp*. Dit staat voor *SSH File Transfer Protocol*, of soms het *Secure File Transfer Protocol*.

Het gebruik van *sftp* lijkt erg op dat van *scp*.

```
1| sftp kevin@slack.host.nl
2| kevin@slackbak's password:
3| Connected to slack.icyx.nl.
4| sftp>
```

Er zijn echter wat kleine verschillen. *sftp* stelt een gebruiker bijvoorbeeld ook in staat om een listing van een map op te vragen. Dit is iets wat in *ftp*

ook mogelijk is, maar wat met *scp* niet kan zonder eerst een shell te openen.

Een ander klein praktisch voordeel boven *ftp* is de mogelijkheid tot het bewaren van de originele timestamps van bestanden wanneer het word verstuurt. Voor de preciese verschillen word er zoals gewoonlijk weer naar de man page verwezen.

Hoofdstuk 10

X

Om op een *Linux* systeem gebruik te kunnen maken van een grafische omgeving is het *X framework* ontwikkeld. Het is een product wat wordt geleid door de *X.Org Foundation*.

10.1 Geschiedenis

De *X.Org foundation* is ontstaan uit een samenwerkingsverband tussen de organisatie die de *X* standaarden regelt en een groep vroegere ontwikkelaars van het *XFree86* project.

De reden dat deze ontwikkelaars zijn overgestapt naar de *X.Org foundation* is een ruzie over het nieuwe licentiemodel van *XFree86*. Er heeft daarom een *fork* plaatsgevonden waarna het project *X11* werd ondergebracht bij de *X.Org foundation*.

10.2 Implementatie

De basis van het *X* protocol is in de jaren 80 ontwikkeld. De bedoeling was om een generiek *client-server* model te hebben om grafische toepassingen te gebruiken. Er moest op een willekeurige machine een applicatie gedraaid moeten worden, die gebruikt kon worden op een willekeurige andere machine. De keuze van het platform zou dit niet mogen belemmeren.

Er zijn binnen het *X* protocol twee partijen. Deze hebben allebij een eigen taak.

10.2.1 X server

De *X server* verzorgt de faciliteiten om de verschillende *X clients* te kunnen laten communiceren. Ook biedt het een *display* aan. Dit betekent dus dat er op een *client* een *X server* draait.

10.2.2 X client

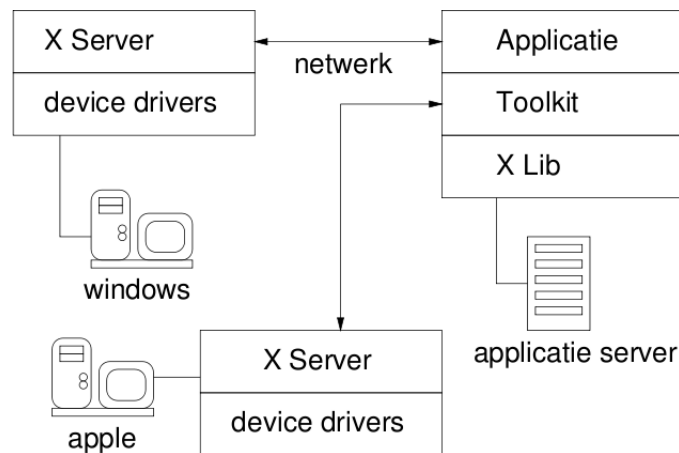
De *X client* zal de diensten van de *X server* gebruiken. De applicaties zelf zijn dus effectief de *X clients*. Als er op een applicatie een grafisch programma wordt gestart zal dit dus een *client* zijn naar de *X server* van de verbonden gebruiker.

10.2.3 Protocol

Voor de communicatie tussen de *X server* en de *X client* is er gekozen voor een compleet transparant protocol. Het is niet afhankelijk van het medium waar het transport over plaatsvindt, waardoor gebruikers ook via het netwerk software kunnen gebruiken.

10.2.4 Communicatie

Wanneer alle bovenstaande aspecten bij elkaar worden gevoegd zal er het volgende totaalbeeld uit komen, zie figuur 10.1:



Figuur 10.1: X client/server overzicht

Hier is duidelijk te zien hoe de verschillende onderdelen van *X* met elkaar samenwerken. Een applicatie gestart op een applicatie server is generiek. Deze zal verbinding maken met de *X server* van de *client*, welke de hardware-specifieke zaken regelt. Om deze reden zal de “Apple” computer dus een andere *X server* hebben dan de “Windows” computer.

10.3 Window Manager

Een *window manager* is een pakket wat alle zaken regelt die gerelateerd zijn aan het venster beheer. Denk hierbij bijvoorbeeld aan de mogelijkheid om vensters te verslepen of van grootte te veranderen. De meeste *window managers* zullen ook voor een *desktop* omgeving zorgen, waardoor het pakketten zijn die sterk zijn geïntegreerd met basis *desktop* applicaties.

De bekendste toepassing die een *window manager* uitvoert is het *reparenten* van een scherm. Hier wordt er nog een extra “venster” om een programma heen gezet, welke vaak bestaat uit de titelbalk. Dit kan dan bijvoorbeeld worden gebruikt om vensters te verslepen.

10.4 Installatie

Op *Slackware* zijn er ook *packages* van *Xorg* beschikbaar. Het installeren hiervan is dan ook relatief eenvoudig.

Er kan gekozen worden om de volledige *X* suite te installeren, of om de minimale delen ervan te kiezen. Omdat het iets gemakkelijker is om de complete *X* suite te installeren zal dat worden gehanteerd in het onderstaande voorbeeld.

```
1 kevin@slackbak:/tmp$ mkdir x
2 kevin@slackbak:/tmp$ cd x; wget -r -A txz -nd ftp://ftp.fu-
   berlin.de/unix/linux/mirrors/slackware/slackware-13.37/
   slackware/x/
3 [...]
4 FINISHED --2010-12-15 17:55:16--
5 Downloaded: 299 files , 84M in 9m 34s (150 KB/s)
6 kevin@slackbak:/tmp/x$ su
7 Password:
8 root@slackbak:/tmp/x$ installpkg *
```

Kies als *url* natuurlijk een *mirror* naar keuze uit. Als *installpkg* klaar is zal de basis *X* omgeving geïnstalleerd zijn.

10.4.1 KDE

Er zijn verschillende *window managers* beschikbaar, maar in dit voorbeeld zal *KDE* worden geïnstalleerd. Deze *window manager* is ook weer in de *Slackware* repositories te vinden, waardoor het erg gemakkelijk is om op het systeem te zetten. De installatie gaat als volgt:

```
1| kevin@slackbak:/tmp$ mkdir kde
2| kevin@slackbak:/tmp$ cd kde; wget -r -A tgz -nd ftp://ftp.fu-
   |   berlin.de/unix/linux/mirrors/slackware/slackware-13.37/
   |   slackware/kde/
3| [...]
4| FINISHED --2010-12-15 13:06:36--
5| Downloaded: 35 files , 374M in 6m 4s (1,03 MB/s)
6| kevin@slackbak:/tmp/kde$ su
7| Password:
8| root@slackbak:/tmp/kde$ installpkg *
```

10.4.2 Fluxbox

Mocht *KDE* te groot bevonden worden kan er ook naar de veel minimalistischere, maar lichtere desktop omgeving is *Fluxbox* gekeken worden. Ook deze is gemakkelijk in gebruik, maar ontzettend flexibel in de configuratie. De installatie gaat als volgt:

```
1| kevin@slackbak:/tmp$ mkdir fluxbox
2| kevin@slackbak:/tmp$ cd fluxbox; wget ftp://ftp.fu-berlin.de/
   |   unix/linux/mirrors/slackware/slackware-13.37/slackware/xap/
   |   fluxbox-1.1.1-i486-2.tgz
3| [...]
4| 2010-12-18 13:27:57 (392 KB/s) - fluxbox-1.1.1-i486-2.tgz
   |   saved [654388]
5| kevin@slackbak:/tmp/fluxbox$ su
6| Password:
7| root@slackbak:/tmp/fluxbox$ installpkg fluxbox-1.1.1-i486-2.tgz
```

10.5 X configureren

Moderne implementaties van *X* hebben autodetectie mogelijkheden aan boord. Dit betekent dat het veel eenvoudiger is geworden om *X* te configureren, er hoeft nu niets meer te worden gedaan.

Mocht het gewenst zijn om een configuratie bestand te genereren, zodat er specifieke *X* instellingen toegepast kunnen worden. Om een basis configuratie bestand te genereren is het mogelijk gebruik te maken van *xorgsetup*. Dit is dus een optionele stap.

```
1|root@slackbak:/$ xorgsetup |
```

De volgende stap is het opgeven van de *window manager*. Dit kan met het commando `xwmconfig`:

```
1|kevin@slackbak:$ xwmconfig |
```

10.6 X Starten

X kan nu gestart worden doort het commando `startx` uit te voeren.

10.7 Remote X

Aangezien het *X11* framework dus de mogelijkheid biedt om applicaties remote te draaien zal er enige uitleg volgen over hoe dit is uit te voeren.

De gemakkelijkste manier is om een *ssh* verbinding te openen met *X forwarding*. Dit kan erg simpel met het programma `ssh`:

```
1|kevin@slackbak:~$ ssh -XY host.nl |
```

De “-X” optie zorgt ervoor dat er aan *X forwarding* gedaan kan worden, met “-Y” wordt gezegd dat de *X11* verbinding te vertrouwen is. Door de laatste optie hoeven er geen specifieke beveiligings opties te worden toegepast, waardoor het erg eenvoudig in gebruik is.

Deze optie vereist wel dat de *SSH server* deze mogelijkheden ondersteund. In het onderstaande voorbeeld is dit het geval:

```
1|kevin@slackbak:~$ grep "X11Forwarding" /etc/ssh/sshd_config |
2|X11Forwarding yes |
```

Hoofdstuk 11

Device files

Hardware wordt binnen *Linux* gezien als een speciale file. Wanneer je een device wilt gebruiken, kan je dat gewoon op je filesystem benaderen. Er kan naartoe geschreven worden, maar er kan ook uit gelezen worden. Deze abstractie biedt voordelen omdat het het lezen en schrijven van devices erg simpel kan maken. Voor veel software is het namelijk helemaal niet nodig om op een erg laag niveau met de hardware te communiceren; alles wat ze willen is bijvoorbeeld het lezen van de toets die op je toetsenbord is ingedrukt.

Om het gebruik van deze speciale device files mogelijk te maken word er gebruik gemaakt van speciale device files. Dit zijn inodes die niet verwijzen naar een block op de disk, maar juist verwijzen naar een device. Dit word gedaan door naar het *major* en *minor* number van een device te wijzen.

Een *major* device nummer is gerelateerd aan het type device. Deze zijn statisch in de kernel opgenomen[16]. Wanneer we naar deze lijst kijken kunnen we bijvoorbeeld zijn dat de eerste *tty* het *major* device id 4 krijgt. Daarnaast krijgt hij *minor* device id het getal 1.

```
1 | 4 char          TTY devices
2 |                 0 = /dev/tty0          Current virtual console
3 |                 1 = /dev/tty1          First virtual console
4 |                 ...
5 |                 63 = /dev/tty63        63rd virtual console
```

Hieruit kunnen we dus opmaken dat `/dev/tty0` een inode zou moeten hebben die verwijst naar *major device* 4 en *minor device* 1. Dit kunnen we controleren met *stat*.

```
1 | root@slackbak:/# stat --printf="Inode:\t\t\t%i\nMajor:Minor
   |      device:\t\t:%T\nSize in bytes:\t\t%s\n" /dev/tty1
2 | Inode:                1980
3 | Major:Minor device:  4:1
4 | Size in bytes:       0
```

11.1 Device types

Er zijn twee hoofdgroepen van speciale devices. De meeste zullen behoren tot de groep *character devices*.

11.1.1 Character devices

Deze groep staat ook bekend onder de naam *raw devices*. Dit komt omdat je ook directe toegang krijgt tot een device. Het is dus mogelijk om alle lees-, en schrijf operaties moeten uitvoeren in de aard van het device.

Een virtuele terminal zal bijvoorbeeld gelezen en geschreven kunnen worden per karakter. Het zijn immers textuele “apparaten”. Er kan dus op een simpele manier tekst van en naar een apparaat gestuurd worden.

11.1.2 Block devices

Wanneer we het over een harde schijf gaan hebben zal dit anders worden, dan zullen we in hele sectoren gaan moeten lezen en schrijven. Dit kan handig zijn voor gebruik in combinatie met `dd`, maar het is iets wat je meestal niet wil. Wanneer je een paar bytes van een file wilt lezen is het natuurlijk onzinnig om de complete sector te moeten uitlezen.

Om dit gebruik te kunnen cachen, en daarmee veel schaalbaarder te kunnen maken voor de multi user omgeving die *Linux* meestal is, zijn er voor onder andere schijven zogenaamde *block devices*. Dit is een extra laag over het ruwe character device heen die het systeem in staat stelt om te cachen. Veel gebruikte blocks kunnen hierdoor dus in het werkgeheugen van een systeem blijven, wat de performance ten goede komt.

Omdat er buffers worden gebruikt heeft het wel als nadeel dat er data verloren kan gaan. Een systeem wat uitgaat voordat de buffers zijn *geflusht* zal alle data in buffers kwijt zijn. Dit kan verholpen worden door de system call `flush()`, die de buffers zal synchroniseren met het onderliggende systeem.

11.2 Belangrijke files

Omdat sommige device files erg veel gebruikt worden, of erg handige features hebben, zullen we een paar van de meest gebruikte devices uit te leggen. Er zal gekeken worden naar hoe ze gebruikt kunnen worden, en waar eventueel op gelet moet worden.

11.2.1 null

Dit device wordt ook wel de digitale prullenbak genoemd. Het *null* device geeft een gebruiker namelijk een oneindig groot “zwart gat”, wat alles weggooid wat er aan hem gegeven wordt. Het zal ook nooit output genereren.

Dit kan bijvoorbeeld erg handig zijn wanneer de output van een script stil moet zijn. Dit is iets wat vaak voorkomt wanneer een script als *cron* word ingesteld. Er kan bijvoorbeeld gekozen worden om de standaard output door te sturen naar *null*, maar om de error stream gewoon zijn gang te laten gaan.

```
1| kevin@slackbak:~$ mijnscrip.sh 1> /dev/null
```

Wanneer er gelezen word van *null* zal er een *EOF* gereturned worden.

11.2.2 zero

Het *zero* device zal, wanneer er uit gelezen wordt, alleen maar *NULL* bytes (`\0`) geretourneerd. Wanneer er naartoe geschreven wordt vertoond het hetzelfde gedrag als *null*.

11.2.3 random

Het *random* device zal willekeurige data geven aan het programma wat hieruit leest. Omdat het de bedoeling is dat dit device fungeert als een echte random number generator, zal er geen data uitkomen als er te weinig entropie is. Dit betekend dat een `read()` call dus kan blokken tot er genoeg entropie is om aan het lezende proces te geven.

Dit device dient gebruikt te worden bij toepassingen die hoge mate van entropie vereisen.

11.2.4 urandom

Het *urandom* device doet hetzelfde als *random* maar zal *niet* blocking zijn. Het betekend wel dat er een concessie wordt gedaan op het gebied van entropie. Wanneer de interne entropie pool van het device (de kernel) leeg is, zal de pool hergebruikt worden. De output krijgt dus minder entropie.

Dit betekend dus in theorie dat het gebruik van *urandom* minder veilig kan zijn als het gebruik van *random*. Uiteraard hangt dit af van de functie waarvoor *urandom* of *random* gebruikt gaat worden.

Hoofdstuk 12

Basiscommando's

Dit hoofdstuk geeft de lezer een overzicht van de meest handige *tools* binnen een *Linux* systeem. Dit overzicht dient meer als “basispakket”, zodat de lezer zichzelf kan redden met het basis gereedschap van het systeem, en is derhalve geen compleet overzicht.

Er zal worden benadrukt wat de taak van het commando is, gevolgd door een praktijk voorbeeldje. Voor ieder commando geldt echter wel dat er meer informatie te vinden is in de bijbehorende *man page*. Deze is op te vragen met het commando `man`. Er zal in de *man page* een compleet overzicht over het commando te vinden zijn.

Mocht het opgezochte commando niet precies voldoen aan de wensen kan er ook altijd nog naar de paragraaf *SEE ALSO* van de *man page* worden gekeken. Hier kunnen eventuele andere commando's worden aangeraden die de gewenste taak wel kunnen volbrengen.

12.1 `awk`

`awk` is een scripttaal specifiek voor *UNIX* en is bedoeld voor het verwerken van tekstbestanden. `awk` wordt veel gebruikt voor kleine automatische tekst bewerkingen binnen zowel *UNIX* als *Linux*. Het gaat te ver om hier helemaal uit te leggen wat `awk` is, meer informatie is te vinden in de *man pages* over `awk`. Het is een taal die erg gericht is op *reguliere expressies*. Mede om deze reden is `perl` gebaseerd op `awk`.

12.2 bzip2

Dit commando biedt een interface om gebruik te maken van een *compressie algoritme*, waardoor het bestanden kan comprimeren. Het gebruikte algoritme heet ook *bzip2*, wat de naam van het commando natuurlijk meteen verklaard.

Een bestand comprimeren is erg gemakkelijk:

```
1| kevin@iusaaset:/tmp$ ls -lh video.dump
2| -rw----- 1 kevin kevin 11M Dec 18 12:50 video.dump
3| kevin@iusaaset:/tmp$ bzip2 video.dump
4| kevin@iusaaset:/tmp$ ls -lh video.dump
5| -rw----- 1 kevin kevin 9.2M Dec 18 12:50 video.dump.bz2
```

Hier is te zien dat het bestand aanzienlijk kleiner is geworden. De standaard extensie van een bestand wat met *bzip2* is gecomprimeerd is *bz2*. Zo'n bestand kan ook weer worden uitgepakt:

```
1| -rw----- 1 kevin kevin 9.2M Dec 18 12:50 video.dump.bz2
2| kevin@iusaaset:/tmp$ bzip2 -d video.dump.bz2
3| kevin@iusaaset:/tmp$ ls -lh video.dump
4| -rw----- 1 kevin kevin 11M Dec 18 12:50 video.dump
```

Nu is het bestand weer in originele staat teruggebracht.

12.3 cat

Leest een bestand en print de inhoud op *STDOUT*, de standaard output. Dit wordt bijvoorbeeld gebruikt om een bestand te lezen:

```
1| kevin@slackbak:~/ $ cat /etc/passwd
2| [...]
```

12.4 cron

Soms is het handig om een bepaald script of een bepaald commando dagelijks of wekelijks op een bepaald tijdstip te laten runnen. Dit kan gedaan worden door middel van een *cronjob* via de *crontab*. Een praktisch voorbeeld is het configureren van nachtelijke backups. Via `man crontab` is precies te vinden hoe het commando `crontab` werkt.

Om de *cronjobs* van een gebruiker aan te passen volstaat het om als die gebruiker het commando `crontab` te starten:

```
1| paul@slackbak:~ $ crontab -e
```

De standaard editor zal nu worden geopend, waarna er instellingen gedaan kunnen worden om een *cron* opdracht aan te maken. Het volgende voorbeeld zal elk uur om 10 minuten na het hele uur een bestand verwijderen:

```
1| 10 * * * * rm /tmp/tijdelijk_bestand
```

De *** staat voor een wildcard, waardoor deze altijd wordt uitgevoerd. De volgorde van de opties is: minuten, uren, dagen, maanden en dag van de week. Op deze manier kan er heel precies bepaald worden wanneer het script wordt uitgevoerd.

Het kan natuurlijk ook voorkomen dat een bepaalde opdracht enkele keren per uur uitgevoerd moet worden. 3 x per uur, dus iedere 20 minuten, is bijvoorbeeld te bereiken door de volgende syntax:

```
1| */3 * * * * rm /tmp/tijdelijk_bestand_2
```

Zie uiteraard voor meer informatie de *man pages*.

12.5 diff

Het commando *diff* wordt gebruikt om de verschillen tussen twee bestanden te analyseren. Dit wordt gedaan door de bestanden regel voor regel te vergelijken.

```
1| kevin@slackbak:~$ diff passwd passwd.gewijzigd
2| 26,27c26,27
3| < kevin:x:1000:100:Kevin van der Vlist , , ,:/home/kevin:/bin/bash
4| < paul:x:1001:100: , , ,:/home/paul:/bin/bash
5| ---
6| > kevin:x:1000:100:Kevin van der Vlist , , ,:/home/kevin:/bin/dash
7| > paul:x:1001:101: , , ,:/home/paul:/bin/bash
```

Hier is te zien dat er op regel 26 en 27 wijzigingen zijn.

Het programma kan ook worden gebruikt om een *patch* te creëren:

```
1| kevin@slackbak:~$ diff -u perf.c perf_veranderd.c
2| --- perf.c      2010-12-13 20:07:41.441979269 +0100
3| +++ perf_veranderd.c  2010-12-13 20:07:27.469135267 +0100
4| @@ -301,6 +301,7 @@
5|         { "sched",      cmd_sched,      0 },
6|         { "probe",     cmd_probe,     0 },
7|         { "kmem",      cmd_kmem,      0 },
8| +         { "extra",    cmd_extra,    0 },
9|     };
10|     unsigned int i;
11|     static const char ext[] = STRIP_EXTENSION;
```

Om het bestand `perf.c` op een ander systeem op exact dezelfde manier te wijzigen hoeft alleen de *patch* te worden verstuurd. Deze methode wordt veel gebruikt bij software ontwikkeling, omdat dan niet alle aangepaste bestanden opgestuurd hoeven te worden.

12.6 file

Dit programma kan worden gebruikt om het bestandstype te analyseren:

```
1| kevin@slackbak:~$ file hoofdstuk01.tex
2| hoofdstuk01.tex: LaTeX document text
```

12.7 find

Dit is één van de programma's die onmisbaar is voor een gebruiker van een systeem. Het programma `find` geeft legio mogelijkheden om te zoeken. er zullen een paar handige scenario's geschetst worden.

Zoek alle bestanden in `/etc/`, met `sl??rc` in hun naam, en list ze met `ls`:

```
1| kevin@slackbak:~$ find /etc/ -name "sl??rc" -type f -ls
2| 130562 24 -rw-r--r-- 1 root root 21599 May 19 2010 /etc/slrn.rc
3| 130332 4 v-rw-r--r-- 1 root root 1437 Feb 19 2010 /etc/slsh.rc
```

Zoek alle bestanden in `/usr/bin`, die beginnen met `ps`, groter zijn dan `5k`, maar kleiner dan `10k`:

```
1| kevin@slackbak:~$ find /usr/bin/ -name "ps*" -size +5k -a -size
   -10k
2| /usr/bin/psc
3| /usr/bin/psidtopgm
4| /usr/bin/psset
5| /usr/bin/psmandup
```

Als laatste zoeken we in `/var/log`, behalve `/var/log/packages`, naar bestanden waarin het woord *swap* voorkomt. Daarna printen we de file en de regel met het woord:

```
1| kevin@slackbak:~$ find /var/log -wholename /var/log/packages -
   prune -o -name dmesg -exec grep "swap" '{}' /dev/null \; -
   print
2| /var/log/dmesg: Adding 3140700k swap on /dev/sda3. Priority:-1
   extents:1 across:3140700k
```

12.8 fsck

`fsck` controleert en repareert indien nodig het filesystem. Wanneer er een fout op het filesystem aanwezig is kan `fsck` dit in veel gevallen repareren. `fsck` draait standaard om de X mounts om ervoor te zorgen dat het filesystem niet beschadigd is.

12.9 grep

`grep` is één van de meest gebruikte commando's in dit dictaat. Het is een zeer krachtig commando, omdat het gemakkelijk streams kan filteren op patronen waar een gebruiker in geïnteresseerd is. De naam *grep* komt van:

global / regular expression / print.

[22] Doordat `grep` zo'n krachtig commando is kan het ontzettend veelzijdig ingezet worden. Met de verschillende opties kan `grep` ook gemakkelijk aangepast worden voor specifieke doeleinden. Hiervoor wordt ook weer naar de *man pages* van `grep` verwezen.

Wat `grep` eigenlijk doet is een bestand doorzoeken met een opgegeven *reguliere expressie*. Juist door het gebruik van deze *reguliere expressies* en de grote hoeveelheid parameters van `grep` is het zeer krachtig. Voor een voorbeeld van `grep` kunnen we naar bijna ieder hoofdstuk in dit dictaat verwijzen.

12.10 gzip

Een ander programma om mee te *comprimeren* is `gzip`. Dit biedt samen met `gunzip` en `zcat` mogelijkheden om data te comprimeren met behulp van het *deflate*[11] algoritme. Het comprimeren van data kan erg eenvoudig:

```
1| kevin@iusaaset:/tmp$ ls -lh dictaat.tex
2| -rw-r--r-- 1 kevin kevin 8.6K Dec 19 19:47 dictaat.tex
3| kevin@iusaaset:/tmp$ gzip dictaat.tex
4| kevin@iusaaset:/tmp$ ls -lh dictaat.tex.gz
5| -rw-r--r-- 1 kevin kevin 3.7K Dec 19 19:47 dictaat.tex.gz
```

Om de inhoud van een bestand te bekijken wat verkleind is met `gzip` kan gebruik gemaakt worden van `zcat`. Dit werkt exact hetzelfde als `cat`, maar leest data uit een *gzip'd* bestand. Dit laat het originele bestand gecomprimeerd zien:

```
1| kevin@iusaaset:/tmp$ zcat dictaat.tex.gz
2| \documentclass[a4paper,11pt]{report}
3| \usepackage{graphicx}
4| [...]
```

Er kan ook worden gekozen om de data uit te pakken. Dit is ook heel eenvoudig met het commando **gunzip**.

```
1| kevin@iusaaset:/tmp$ ls -lh dictaat.tex.gz
2| -rw-r--r-- 1 kevin kevin 3.7K Dec 19 19:47 dictaat.tex.gz
3| kevin@iusaaset:/tmp$ gunzip dictaat.tex.gz
4| kevin@iusaaset:/tmp$ ls -lh dictaat.tex
5| -rw-r--r-- 1 kevin kevin 8.6K Dec 19 19:47 dictaat.tex
```

Ook met bijvoorbeeld **gzip -d** kan je een gzip file uitpakken.

12.11 ldd

Dit programma kan worden gebruikt om na te gaan van welke shared libraries een bestand afhankelijk is. Dit is handig om te controleren waarom een bepaald programma niet meer kan starten:

```
1| kevin@slackbak:~$ ldd /usr/bin/echo
2|          linux-gate.so.1 => (0xffffe000)
3|          libc.so.6 => /lib/libc.so.6 (0xb76aa000)
4|          /lib/ld-linux.so.2 (0xb7823000)
```

12.12 less

Het programma **less** is de uitgebreidere variant van **more**. De *man page* beschrijft het als *less - opposite of more*. Je kan hier mee door teksten bladeren, zoeken naar woorden of andere dingen. Dit programma is aan te raden bij het lezen van teksten en zal standaard ook door **man** worden gebruikt.

12.13 ln

Met het programma **ln** kunnen links worden aangemaakt worden om naar bestanden te verwijzen. Zo kan bijvoorbeeld een binary meerdere namen hebben, maar toch maar een keer opgeslagen zijn.

Er zijn wel verschillen tussen hard-, en soft links. Hard links laten een bestand naar dezelfde *inode* verwijzen, terwijl soft links gebruik maken word van relatieve pad verwijzingen.

Om het verschil duidelijk te maken hebben we een klein voorbeeldje.

```
1| kevin@slackbak:~$ touch bestand
2| kevin@slackbak:~$ ln bestand hardlink
3| kevin@slackbak:~$ ln -s bestand softlink
4| kevin@slackbak:~$ ls -li
5| total 8
6| 393379 -rw-r--r-- 2 kevin users 0 2010-12-13 19:57 bestand
7| 393379 -rw-r--r-- 2 kevin users 0 2010-12-13 19:57 hardlink
8| 393380 lrwxrwxrwx 1 kevin users 7 2010-12-13 19:57 softlink ->
   |     bestand
```

Hier is te zien dat *softlink* een echte link is naar de file *bestand*. De file *hardlink* heeft dezelfde *inode* als de file *bestand*.

12.14 man

Voor dit commando willen we terugverwijzen naar de eerdere uitleg over het commando `man` in paragraaf 4.7.

12.15 more

`more` is een programma wat tekst opvult over het hele scherm. Hierdoor kan het worden gebruikt om gemakkelijk grote teksten door te lopen. Het kan echter niet terug, dus er kan alleen vooruit worden gebladerd.

12.16 nice

Omdat *Linux* een multi-user systeem is, zullen er regelmatig verschillende *processen* met verschillende eigenaren actief zijn. Het kan echter voorkomen dat een *proces* belangrijker is dan gemiddeld.

Om in deze situatie een oplossing te bieden kan er gebruik gemaakt worden van `nice`. Hiermee kan een *proces* een hogere prioriteit gegeven worden. Dit houdt in dat het vaker “aan de beurt is” volgens de *scheduler*.

Standaard heeft een *proces* een *niceness* van 10. Er kan met `nice` echter een waarde tussen -20 (belangrijkst) en de 19 (onbelangrijkst) toegekend worden. Op de volgende manier krijgt `tar` dus een hogere prioriteit:

```
1| root@slackbak:/$ nice -10 tar czf /backup-home.tar.gz /home
```

12.17 patch

Het commando `patch` kan worden gebruikt om een bestand met verschillen, bijvoorbeeld gemaakt met `diff` toe te passen op een bestand:

```
1| kevin@slackbak:~$ patch -p0 < perf.patch
2| patching file perf.c
```

12.18 sed

`sed` staat voor *Stream Editor* en kan tekst streams modificeren. Deze streams kunnen van de *standaard input* komen, maar het kan ook tekst uit een file zijn.

Eén van de meest uitgebreide mogelijkheden om tekst mee te filteren is met *reguliere expressies*[27][8]. Er zal hier niet op ingegaan worden, we verwijzen hiervoor naar de betreffende vakken.

Een voorbeeld van `sed` is het printen van alle usernames uit `/etc/passwd`. De bovenste regel is een voorbeeld regel uit het bestand, de onderste is de output na het gebruik van `sed`:

```
1| root@iusaaset:/$ head -1 /etc/passwd
2| root:x:0:0:root:/root:/bin/bash
3| root@iusaaset:/$ sed 's/\([^:]*\)*/\1/' /etc/passwd | head -1
4| root
```

12.19 stat

Dit commando laat uitgebreide informatie over een bestand zien.

```
1| kevin@slackbak:~$ stat /etc/passwd
2|  File: '/etc/passwd'
3|  Size: 1107      Blocks: 8          IO Block: 4096
   regular file
4| Device: 801h/2049d    Inode: 130608      Links: 1
5| Access: (0644/-rw-r--r--)  Uid: (   0/   root)   Gid: (   0/   root)
6| Access: 2010-12-13 13:47:01.348605445 +0100
7| Modify: 2010-12-07 13:29:30.124950561 +0100
8| Change: 2010-12-07 13:29:30.189443446 +0100
```

12.20 tail

Het programma `tail` kan gebruikt worden om het laatste gedeelte van een bestand te laten zien. Standaard zijn dit 10 regels, maar dit is eenvoudig aan te passen.

Een handige optie van `tail` is “-f”, waarmee een bestand “gevolgd” kan worden. Hierdoor worden nieuwe entries in een file, zoals bijvoorbeeld een log, meteen weergegeven op *STDOUT*. In een shell script zal men dit niet snel gebruiken, maar gewoon in de console is dit erg handig.

12.21 tar

Dit commando wordt gebruikt om archieven te maken met meerdere bestanden. De archieven zijn ongecomprimeerd en te representeren als een lange lijn met bestanden. Dit komt omdat `tar` bedoeld was om bestanden op te slaan op een *tape drive*.

Tegenwoordig wordt `tar` vaak gebruikt in combinatie met `gzip` of `bzip2`. Zo worden meerdere bestanden in een archief geplaatst, welke vervolgens weer als geheel wordt gecomprimeerd:

```
1| kevin@iusaaset:~/dict$ tar -cjf /tmp/dictaat.tar.bz2 *
2| kevin@iusaaset:~/dict$ ls -lh /tmp/dictaat.tar.bz2
3| -rw-r--r-- 1 kevin kevin 1.3M Dec 19 21:53 /tmp/dictaat.tar.bz2
```

Uitpakken van een archief is erg simpel:

```
1| kevin@iusaaset:/tmp/dictaat$ ls -lh
2| total 1.3M
3| -rw-r--r-- 1 kevin kevin 1.3M Dec 19 21:53 dictaat.tar.bz2
4| kevin@iusaaset:/tmp/dictaat$ tar -xjf dictaat.tar.bz2
5| kevin@iusaaset:/tmp/dictaat$ ls -lh
6| total 1.6M
7| -rw-r--r-- 1 kevin kevin 1.4K Dec 17 17:24 appendix_bios.tex
8| -rw-r--r-- 1 kevin kevin 4.5K Dec 17 17:24 appendix_emacs.tex
9| -rw-r--r-- 1 kevin kevin 1.3M Dec 19 21:53 dictaat.tar.bz2
10| [...]
```

Om gebruik te maken van `gzip` in plaats van `bzip2` kan de “j” optie vervangen worden door een “z”.

12.22 wget

Om bestanden van het internet te kunnen downloaden is er het programma `wget`. Dit programma is ontzettend uitgebreid, het kan zelfs gehanteerd wor-

den om complete websites te kopiëren. Voor de details wordt weer verwezen naar de *man page*.

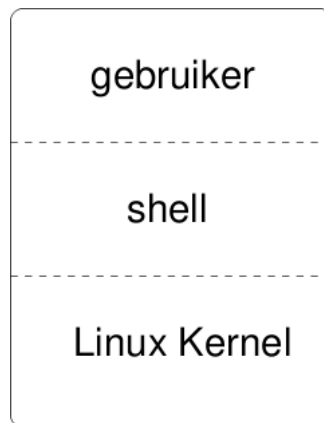
Naast het downloaden via *http* ondersteunt **wget** ook de mogelijkheid om te downloaden via *ftp*:

```
1 kevin@slackbak:~$ wget ftp://ftp.fu-berlin.de/unix/linux/mirrors
  /slackware/slackware64-13.37/source/installer/busybox-1.15.3.
  tar.bz2
2 [...]
3
4 100%[=====>] 1,987,727
   985K/s   in 2.0s
5
6 2010-12-13 20:20:33 (985 KB/s) - "busybox-1.15.3.tar.bz2" saved
  [1987727]
```

Hoofdstuk 13

Shell

13.1 Inleiding shell



Figuur 13.1: De basis

De *shell* staat tussen de gebruiker en het systeem in en dient als *opdracht-interpreter*. De *shell* hiermee vergelijkbaar met *command.com* onder *DOS*. Wanneer een gebruiker een commando geeft controleert de *shell* of dit opgegeven commando correct is. Hierna zal de *shell* de opdracht uitvoeren, of een foutmelding teruggeven naar de gebruiker. De *shell* is hiermee de eerste laag in het besturings systeem.

Op elk systeem is er een *shell* die ervoor zorgt dat commando's worden uitgevoerd. Onder *Linux* zijn er meerdere *shells* zoals bijvoorbeeld de *Bash-Shell*, *Korn-shell* en de *C-shell*.

De *Bourne shell* is de *shell* die origineel op de *AT&T UNIX* systemen stond onder de naam *sh*. *Bourne Again Shell* (**bash**) is de opensource versie en is compatibel met het origineel. Deze *shell* heeft echter wel extra mogelijkheden¹.

Naast *Bash* zijn er tegenwoordig ook nog andere afgeleiden. Een goed voorbeeld hiervan is **dash**, *Debian Almquist Shell*. **Dash** komt af van de in *netBSD* bekende *Almquist Shell* en was naar *Linux* geport in 2002 [19]. Het voordeel van **Dash** is dat het een stuk sneller is als *Bash*, maar toch nog werkt volgens de *POSIX* standard. Tegenwoordig maken bijvoorbeeld *Debian*² en *Ubuntu* gebruik van **Dash**.

Een *Linux* systeem is geen systeem zonder een *shell*, en de scripting mogelijkheden die daardoor aanwezig zijn. Bijna alles is opgebouwd rond de *shell*. Een goed voorbeeld hiervan zijn de opstart scripts in *Slackware*³. Hieronder is een voorbeeld *init* script van de *OpenSSH Server*:

```
1 root@slackje:/etc/rc.d# cat rc.sshd
2 #!/bin/sh
3 # Start/stop/restart the secure shell server:
4
5 sshd_start() {
6     # Create host keys if needed.
7     if [ ! -r /etc/ssh/ssh_host_key ]; then
8         /usr/bin/ssh-keygen -t rsa1 -f /etc/ssh/ssh_host_key -N ''
9     fi
10    if [ ! -f /etc/ssh/ssh_host_dsa_key ]; then
11        /usr/bin/ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key -N
12        ,,
13    fi
14    if [ ! -f /etc/ssh/ssh_host_rsa_key ]; then
15        /usr/bin/ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key -N
16        ,,
17    fi
18    /usr/sbin/sshd
19 }
20
21 sshd_stop() {
22     killall sshd
23 }
24
25 sshd_restart() {
26     if [ -r /var/run/sshd.pid ]; then
27         echo "WARNING: killing listener process only. To kill every
28             sshd process, you must"
29         echo "          use 'rc.sshd stop'. 'rc.sshd restart' kills
30             only the parent sshd to"
```

¹Bourne Again SHell, de hoofdletters "hell" slaan op de problemen die sommige ermee hebben

²*Debian* maakt vanaf de *Debian Squeeze* release standaard gebruik van *Dash*.

³Dit geldt natuurlijk ook voor andere distributies

```

27     echo "          allow an admin logged in through sshd to use
      'rc.sshd restart' without"
28     echo "          being cut off.  If sshd has been upgraded,
      new connections will now"
29     echo "          use the new version, which should be a safe
      enough approach."
30     kill `cat /var/run/sshd.pid`
31 else
32     echo "WARNING: There does not appear to be a parent instance
      of sshd running."
33     echo "          If you really want to kill all running
      instances of sshd (including"
34     echo "          any sessions currently in use), run '/etc/rc.
      d/rc.sshd stop' instead."
35     exit 1
36 fi
37 sleep 1
38 sshd_start
39 }
40
41 case "$1" in
42 'start ')
43     sshd_start
44     ;;
45 'stop ')
46     sshd_stop
47     ;;
48 'restart ')
49     sshd_restart
50     ;;
51 *)
52     echo "usage $0 start|stop|restart"
53 esac

```

Het bovenstaande script zal waarschijnlijk niet helemaal direct begrijpbaar zijn. Wel zullen de grote lijnen al zichtbaar zijn. In de volgende twee hoofdstukken zal de *shell* behandeld worden, waarna het bovenstaande script duidelijk te begrijpen is. Ook zal het maken van eigen scripts hierna mogelijk zijn.

Shell scripting is een van de sterkste punten van *Linux*, omdat het gebruikers en beheerders in staat stelt een systeem ontzettend flexibel te houden. Ook kan het laagdrempelig aan worden gepast naar eigen, specifieke wensen.

13.2 De theorie

Voordat er uitgebreid begonnen wordt met het bespreken van de verschillende mogelijkheden in de *shell* is het belangrijker om te begrijpen hoe de *shell* precies werkt en wat de eisen zijn. Een uitstekende documentatie over

de *shell* is te vinden via `man bash`. Hierin is een goede uitleg te vinden over de werking van de *shell*. Het is mogelijk om uitgebreidere documentatie gerelateerd aan scripting op te zoeken. Een goede plek hiervoor is *tldp*[6], een compleet overzicht van de functionaliteit van de *Bash shell*.

De meeste generieke beschrijving van een *shell* is als volgt:

any program that users employ to type commands

In *UNIX* zijn er verschillende soorten *Shells*. Wanneer een gebruiker inlogt in het systeem zal de *shell* automatisch worden gestart. De meeste *Shells* zijn hiervoor special ontwikkeld. Dit programma heet een *Shell* omdat het het onderliggende operating system verbergt achter de interface van de *Shell*. De *Shell* beheert de technische details van de kernel interface, welke de binnenste laag is binnen elk operating system.

Ook de grafische interfaces, zoals *Gnome*, *KDE* of *Fluxbox* zijn *Shells*. Ze worden ook wel *visuele shells* of grafische *shells* genoemd. [30]

13.2.1 Interactive command line en scripting program language?

Toen de *UNIX shell* voor het eerst uitkwam was dit een bijzondere *shell*. Er waren al wel andere *Shells*, maar de *Shells* welke er al wel waren waren niet en een *Interactive Commandline* en een *Scripting Program Language*. De *UNIX shell* was dit wel, met alle voordelen van dien. Je kon nu zelf programma's draaien met de *Shell* en je had dus een stuk meer vrijheid om dingen te doen.

13.2.2 Shebang

Wanneer een programma gestart wordt moet de file loader van de *Shell* weten waarmee het programma geopend moet worden. Standaard zal dit meestal het programma met de *Shell* waarin op dat moment een gebruiker actief is.

Het kan voorkomen dat het script met een andere *shell* of binary gestart zal moeten worden. Een andere mogelijkheid is het verkrijgen van een garantie dat een bepaalde *shell* het script start. Om dit doel te bereiken kan er gebruik gemaakt worden van een *shebang*, een speciale eerste regel in een script. Deze *shebang* ziet er als volgt uit:

```
1|#!/bin/bash
```

Het hekje met het uitroepteken is de *shebang*. Doordat het hekje commentaar is, zal het genegeerd worden door de interpreter. De program loader zal het echter wel lezen en de goede interpreter aanroepen. In het geval van hierboven dus de *Bash shell*. Er kan op deze manier bijvoorbeeld ook eenvoudig een *Perl*, *PHP* of andere interpreter aangeroepen worden in de *shell*.

13.2.3 Dash in Slackware?

De standaard *Shell* in *Slackware* is *Bash*. Het is echter mogelijk een andere *shell* te kiezen. Van de *Dash shell* zijn er in *Slackware* geen standaard pakketten, maar *Ash*⁴ is wel beschikbaar.

Hieronder is te zien hoe een nieuwe gebruiker wordt aangemaakt, welke gebruik zal maken van de *Ash shell*:

```
1 | root@slackje:~# adduser dashtest
2 | Login name for new user: dashtest
3 | User ID ('UID') [ defaults to next available ]:
4 | Initial group [ users ]:
5 | Additional UNIX groups:
6 | [...]
7 | :
8 | Home directory [ /home/dashtest ]
9 | Shell [ /bin/bash ] /bin/ash
10 | Expiry date (YYYY-MM-DD) []:
11 | New account will be created as follows:
12 | _____
13 | Login name.....: dashtest
14 | UID.....: [ Next available ]
15 | Initial group....: users
16 | Additional groups: [ None ]
17 | Home directory...: /home/dashtest
18 | Shell.....: /bin/ash
19 | Expiry date.....: [ Never ]
```

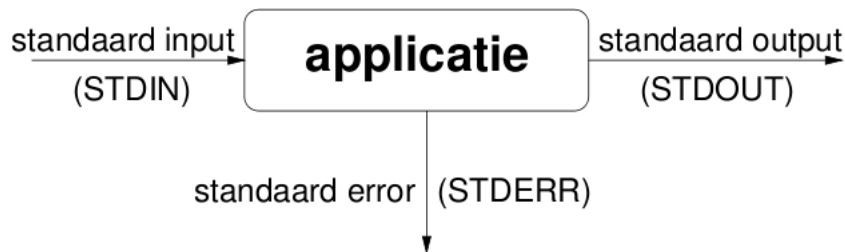
De *Ash shell* uit *Slackware* is niet exact hetzelfde als *Bash* en *Dash*. Na het inloggen zal het eerste verschil al opvallen. Ook zullen dingen als *tab completion* zal niet standaard werken. Andere distributies, zoals *Debian/Ubuntu* zullen standaard instellingen aanbieden waarbij dit wel mogelijk is. Om *Ash* zich hetzelfde te laten gedragen als *Bash* zal er dus wel het een en ander moeten worden aangepast.

In de rest van dit hoofdstuk wordt er vanuit gegaan dat de standaard *shell* de *Bash shell* is. Het gebruik van een andere *shell* is mogelijk, maar hou rekening met de verschillen tussen de *shells*.

⁴Dash is afgeleid van Ash

13.2.4 In en output

De *shell* heeft, net als ieder ander proces, mogelijkheden tot in-, en output van data. De input, *STDIN*, is bijvoorbeeld het toetsenbord⁵ of de muis. De output, *STDOUT*, is bijvoorbeeld het scherm, of een bestand. Hiernaast is er ook nog *STDERR*. Op deze *stream* worden de fouten uitgevoerd. De *STDERR* wordt standaard naar het scherm geschreven, maar kan ook worden doorgestuurd naar iets anders als dit nodig is. Deze *streams* kunnen ook in programma's gebruikt worden. In C kunnen deze streams bijvoorbeeld gewoon met *fopen()* benaderd worden. In 13.2 is een goed voorbeeld te zien van hoe de in/output werkt bij een applicatie. In principe is dit hetzelfde voor alle applicaties.



Figuur 13.2: In en Output

Het kunnen gebruiken van deze streams is heel belangrijk voor *Shells*. Zorg dus dat de basis hiervan goed duidelijk is.

13.2.5 Pipeline

Alle *UNIX* applicaties hebben een standaard input en standaard output mogelijkheid. De *shell* biedt handige functionaliteit bij het doorsluizen van uitvoer met het '|' teken (sluis, engels: pipe). Het '|' teken sluis de uitkomst van de eerste opdracht naar de input van de tweede. Een kort voorbeeld:

```
1| grep kernel /var/log/syslog | more |
```

cat leest een bestand uit, in dit geval de logfile */var/log/syslog*. De output wordt nu gepiped naar het commando *grep* met een *|*. De output wordt nu niet doorgestuurd naar de *stdout*, en zal dus niet worden weergegeven op je scherm. Zodra *grep* de output ontvangt zoekt *grep* naar de tekenreeks die opgegeven is. Dat is in dit geval "kernel". Voordat het nu doorgestuurd wordt naar de *stdout* wordt het eerst nog gepiped via *more*. *more* zorgt

⁵Dit kan ook wat anders zijn, zie de paragraaf Piping

ervoor dat de output gestopt wordt zodra het scherm vol is. Op deze manier kan het resultaat gemakkelijk bekeken worden. Alles gelezen? Dan kan er naar de volgende buffer gegaan worden.

```
1| grep bash /etc/passwd | sort > ~/bash_gebruikers.txt |
```

`grep` ontvangt de inhoud van de file `/etc/passwd` en doorzoekt de inhoud naar de tekenreeks “bash”. Zit die reeks niet in een regel, dan wordt deze genegeerd. Alle regels van `passwd`-bestand met “bash” in de regels worden nu doorgesluisd naar `sort`. Deze plaatst ze op alfabetische volgorde.

Net zoals bij het vorige voorbeeld ontvangt `grep` weer het resultaat van `cat`, waarbij het resultaat de inhoud van een bestand is, in dit geval `/etc/passwd`. Nadat er gezocht is op de string “bash” wordt het gepiped naar `sort`, welke ze sorteert op alfabetische volgorde.

Het `>` teken wordt gebruikt om het resultaat op de *stdout* weg te schrijven. Het resultaat wordt nu naar de file “bash_gebruikers” geschreven. Let wel op dat gebruik maken van `>` betekend dat de file, waar de output naartoe wordt gestuurd, gewist zal worden. Om de output achter de huidige content te plakken kan er gebruik gemaakt worden van `>>`. Wanneer de file niet bestand zal die dan wel gemaakt worden.

Dit soort constructies stellen een gebruiker dus in staat om de uitvoer, *STDOUT* via een *pipeline* als invoer *STDIN* naar het volgende programma te sturen.

STDERR redirecten

Hiervoor was al te zien dat *STDOUT* te redirecten is naar een bestand. Maar wat gebeurt er wanneer er fouten optreden? Die moeten misschien ook wel ergens heen. Ook de data op *STDERR* is te redirecten:

```
1| cat /etc/passwd | grep test 2> grep_errors.txt |
```

In het voorbeeld hierboven is te zien dat er geen gebruik gemaakt wordt van `>` maar van `2>`. Dit houdt in dat de fouten die eventueel optreden worden geschreven naar het bestand “grep_errors.txt”. De gewone output wordt nu naar *STDOUT* geschreven. Deze zal niet worden opgeslagen in het bestand.

```
1| cat /etc/passwd | grep test 1>&2 |
```

In sommige gevallen is het handig om de output van *STDOUT* te redirecten naar *STDERR*. Dat is precies wat de `1>&2` doet.

```
1| cat /etc/passwd | grep test 2>&1 |
```

Het is vaak veel handiger om dit precies omgekeerd te doen, ofwel de *STDERR* naar *STDOUT* te redirecten. En ook hier is iets voor bedacht, en wel `2>&1`

en dus precies het omgekeerde. Let dus goed op de syntax en kies de juiste redirect in de juiste situatie.

```
1|rm tmp/ -rfv &> /dev/null
```

`&>` zal alle output die vanaf `rm` afkomt van zowel *STDOUT* als *STDERR* worden redirect naar het opgegeven bestand. In dit geval wordt dit geredirect naar */dev/null*.

13.2.6 TAB-completion

TAB-completion is eenvoudig en werkt erg makkelijk volgens een simpel principe: Als er genoeg letters ingevoerd zijn zodat een bestand, opdracht, of naam van een directory kan worden achterhaalt dan vult *Bash* de rest aan. Een voorbeeld:

```
1| /usr/lo [tab] wordt /usr/local
```

Als er nu op TAB wordt gedrukt vult *Bash* het aan, omdat het ziet dat er gezocht wordt naar *'/usr/local'*.

Als er bij */usr/li* op tab wordt gedrukt kan *Bash* niet het juiste resultaat vinden omdat er meerdere directories zijn die beginnen met */usr/li*. Als er vervolgens nogmaals op tab wordt gedrukt geeft *Bash* alle mogelijkheden weer.

13.2.7 Opdrachtalliassing

```
1|alias psa='ps -aux | less '  
2|alias cp='cp -i '  
3|alias mv='mv -i '  
4|alias rm='rm -i '
```

Alias geeft de mogelijkheid om een commando te herschrijven met extra opties. Zet het bovenstaande voorbeeld in het bestand *~/.bash_profile* in de home-directory (maak het zonodig aan). Dit bestand zijn extra configuratie instellingen voor de *Bash* shell. Globale *Bash* instellingen moeten in het bestand */etc/profile* gedaan worden. Activeer de aanpassingen door opnieuw in te loggen of via het commando `source ~/.bash_profile`.

13.2.8 Historylist

Bash onthoudt bij een sessie de uitgevoerde opdrachten. Bij het uitloggen van het systeem worden alle opdrachten in de file *~/.bash_history* in de home-directory geschreven. Wanneer een gebruiker de volgende keer inlogt zal door het commando *history* precies te zien zijn wat er bij een vorige sessie

is gedaan. Als er tijdens een sessie een opdracht uitgevoerd moet worden die al uitgevoerd is kan er met het pijltje-omhoog de vorige opdrachten worden geselecteerd.

Een andere functionaliteit die *Bash* biedt is om een commando dat net is ingegeven te herhalen. Hiervan een korte beschouwing:

```
1| mount /dev/hdd -t iso9660 -o ro /mnt/cdrom
```

Omdat dit soort lange opdrachten veel tijd kosten om in te tikken is het gemakkelijk als zo'n opdracht herhaalt moet worden. Door "`!<commando>`" te voer je de vorige opdracht uit. In dit geval is dat dus '`!mount`'.

13.2.9 Jobcontrol

Jobcontrol is mogelijk in *Linux* omdat het een multi-tasking operating system is. Met job control kan in een *shell* meerdere programma's tegelijkertijd worden uitgevoerd. Normaal gesproken draait een opdracht die wordt uitgevoerd op de voorgrond. Met andere worden, de prompt verschijnt pas weer als de opdracht is beëindigd. De makkelijkstse manier om een job op de achtergrond te plaatsen is door aan het einde van de opdracht een "&" (ampersand) te plaatsen.

De volgende opdracht zoekt vanaf de root ('/') naar alle bestanden die op '.txt' eindigen en zet het resultaat ervan in het bestand 'tekstbestanden'. Doordat er een ampersand achterstaat wordt het op de achtergrond uitgevoerd en verschijnt de *shell* prompt weer. `ls -IR /` laat alle bestanden vanaf root zien.

Een op de voorgrond gestart programma kan naar de achtergrond worden gestuurd door *ctrl-z*.

```
1| find / -name '*.txt*' > tekstbestanden &
2| ls -IR / &
```

Met de opdracht worden alle achtergrond-processen weergeven en in dit geval zou het dus de find opdracht weergeven en de ls opdracht.

```
1|[1]- Running find / - name '*.txt *' > tekstbestanden &
2|[2]+ Running ls -lR / &
```

Wanneer 'Running' in 'Stopped' veranderd is het process gestopt. Aannemende dat het nog loopt kan het naar de voorgrond gehaald worden met `fg %1` of met `fg find`. Het komt voor dat een lopende opdracht beëindigd moet worden. Met de opdracht `kill` is een opdracht af te breken op het *PID* (process ID). Het PID is te zien door `ps -aux` in te geven. Ook kan *KILL* lopende achtergrond processen stoppen: `kill %2` zal in dit geval `ls`

-IR / beëindigen. Mocht het voorkomen dat alle processen met een bepaalde naam beëindigd moeten worden dan zou dat het beste met `killall <naam>` gedaan kunnen worden.

13.2.10 Patronen, Joker-tekens en Accolades

Een krachtige optie van de *Bash shell* is het gebruik van patronen om bestanden en opdrachten mee aan te duiden.

```
1| \* Staat voor 0 of meerdere willekeurige tekens (wildcard,  
   |   | jokerteken)  
2| \? Staat voor 1 willekeurig teken  
3| [...] Staat voor 1 van de tekens tussen de haken  
4| [A-F] Staat voor alle tekens tussen het eerste en tweede  
5| [!...] Staat voor elk ander teken dan dat er tussen de haken  
   |   | staat  
6| [!A-F] Staat dus voor elk ander dat niet tussen de A en de F  
   |   | zit
```

Alle bestanden van drie letters beginnende met de 'a' en eindigen op de 'z' kunnen gevonden worden met:

```
1| ls -l a?z
```

Alle bestanden die met een 'a' beginnen en eindigen op een 'z' kunnen worden weergegeven met:

```
1| ls -al a*z
```

Stel er moet een lijst komen van alle bestanden beginnende met de 'a', 'b', 'c', of 'd', dan kan je dit doen met:

```
1| ls -l a* b* c* d*
```

Een iets betere oplossing is:

```
1| ls -l [abcd]*
```

Maar omdat abcd opeenvolgende letters zijn kan die het best zo worden uitgevoerd:

```
1| ls -l [a-d]*
```

Accoladen-uitbreiding is een manier om opdrachten te maken ongeacht of het bestand of de opdracht wel bestaat:

```
1| mkdir tijddir1,2,3,4
```

13.2.11 Opdracht-substitutie

Opdracht-substitutie maakt het mogelijk om de opdrachtregel door zijn resultaat te veranderen. De opdracht moet wel tussen *backticks* ‘ worden geplaatst.

```
1| ls -l `find /usr/doc -name '*README*'` |
```

Deze opdracht zoekt naar alle **README** bestanden in de */usr/doc* directory, en zorgt ervoor dat *find* ze één voor één aan *ls -l* doorgeeft. De volgende opdracht werkt niet omdat *ls* de gegevens vanuit het standaard invoer kanaal negeert.

```
1| find /usr/doc/ -name '*README*' | ls -l |
```

Dit is te omzijken door *xargs* toe te voegen:

```
1| find /usr/doc/ -name '*README*' | xargs ls -l |
```

Hoofdstuk 14

Shell scripting

Het beheersen van *UNIX* commando's en een shell script taal zijn het fundament van een goede systeembeheerder en maken zijn leven vele malen aangenamer. We gaan hieronder kort de basis uitleggen van shell scripting. Dit is de pure basis, hierna zal het *SSH* script van het vorige hoofdstuk kunnen begrijpen en waarschijnlijk ook na kunnen maken. De rest van dit hoofdstuk zal gaan over Statements, functies, en structuren. Maar ook over het gebruik van andere programmeer talen, en commando's.

14.1 Statements, expressies en controle structuren

Een belangrijk onderdeel van elke programmeer taal zijn de aanwezige statements. Het lijkt eigenlijk erg op elkaar lijkt als je een normale programmeer taal zult vergelijken met *Bash*, maar het is toch weer niet helemaal hetzelfde. We gaan er een beetje van uit dat je de basis van programmeren wel kent, en ongeveer weet wat het doet.

14.2 Variabelen en het if-statement

Variabelen bieden de mogelijkheid om tijdelijk dingen op te slaan. De *Bash-shell* maakt weinig onderscheid in type variabelen zo bestaan er wel integers en strings, maar zijn er bijvoorbeeld geen doubles of andere *datatypen*.

Zet het onderstaand voorbeeld in het bestand 'script.sh'. Maak het executable door `chmod + x`, en roep het daarna aan met `./script.sh`.

```
1|#!/bin/bash
```

```

2 echo "Hey, wat is je naam?"
3 read naam
4 if [ $naam == $LOGNAME ]; then
5     echo "Ha $naam"
6 else
7     echo "Dat is niet waar!"
8 fi
9 exit

```

Op de eerste regel staat `#!/bin/bash`, dit geeft aan dat dit een shellsript is. Ondanks dat het geen vereiste is om een werkend script te maken wordt het vaak wel gebruikt. Soms wordt bash vervangen door sh.

Het commando `echo` zet iets op het scherm waarna `read` de input van de gebruiker opslaat in de variable 'naam'. Het if-statement vergelijkt de waarde van de variable naam met de waarde in de *variable* `$LOGNAME`. Wanneer ze overeen komen (`==`) dan voert de shell de regel onder het if-statement uit, en anders voert de shell het *else-statement* uit. De variable `LOGNAME` is een systeem-variable die de inlognaam van de user bevat. Als er met de waarde in een variable wordt gewerkt moet er een `$` voor komen.

De structuur van het *if-statement*:

```

1 if conditie 1
2     then
3     statement1
4     statement2
5     .....
6 elsif conditie2
7     then
8     statement3
9     statement4
10    .....
11 elsif conditie3
12     then
13     statement5
14     statement6
15     .....
16 fi

```

14.3 Systeem variabelen

Hierboven maakten we al gebruik van *systeem* variabelen, welke door de *Shell* zelf een waarde gegeven wordt. Naast `$LOGNAME` zijn dit een aantal belangrijke voorbeelden:

- `$UID`: Bevat welke *real* user ID de ingelogde gebruiker is.
- `$DISPLAY`: Bevat welk display er op dat moment op gewerkt wordt.

- *\$PATH*: In dit pad worden de programma's/commando's gezocht welke uitgevoerd worden. Dit moet bijvoorbeeld aangepast worden wanneer je een programma op een ander als standaard locatie installeert.

We kunnen uiteraard niet alle aanwezige *systeem* variabelen hier gaan noemen. Als je op internet of in de *man pages* zoekt kan er nog veel meer voorbeelden vinden als hierboven.

14.4 Expressies

Expressies worden vaak gebruikt bij het testen van *voorwaarden*. Er wordt gekeken of er aan een bepaalde voorwaarde wordt voldaan. Aan de hand van de uitkomst van de voorwaarde wordt er een stuk code uitgevoerd.

14.4.1 String expressies

Expressie	Waar als...
-z string	de lengte van de string 0 is
-n string	de lengte van string niet 0 is
string1 == string2	de twee strings gelijk zijn
string1 != string2	de twee strings ongelijk zijn
string	de string niet NULL is

14.4.2 Integer expressies

expressie	waar als...
int1 -eq int2	de eerste integer gelijk is aan de tweede integer (equal)
int1 -ne int2	de eerste integer ongelijk is aan de tweede integer (not equal)
int1 -gt int2	de eerste integer groter is als de tweede integer (greater)
int1 -ge int2	de eerste integer groter of gelijk is aan de tweede integer (greater or equal)
int1 -lt int2	de eerste integer lager is als de tweede integer
int1 -le int2	de eerste integer lager of gelijk is aan de tweede integer

14.4.3 Bestands expressies

Expressie	Waar als...
-e file	file bestaat
-r file	file bestaat en is readable
-w file	file bestaat en is writeable
-x file	file bestaat en is executable
-f file	file bestaat en is een gewoon bestand
-d file	file bestaat en is een directory
-h file	file bestaat en is een symbolic link
-c file	file bestaat en is een character special file
-b file	file bestaat en is een block special file
-p file	file bestaat en is een named pipe
-u file	file bestaat en is setuid
-g file	file bestaat en is setgid
-k file	file bestaat en heeft een sticky bit
-s file	file bestaat en de grote is meer als 0

Een voorbeeld van een expressie is als volgt:

```
1 | if [ -e /tmp/test/ ]; then
2 |     echo ‘‘/tmp/test/ bestaat!’’
3 | fi
```

14.4.4 Logische operatoren

Expressie	Doel
!	neem het tegenovergestelde van het resultaat van de expressie
-a	AND operator
-o	OR operator

14.5 De shell parameter variabelen

Variabele	Doel
\$0	Naam van het script
\$1 tot \$9	Het eerste tot de negende parameter
\$#	Het totaal aantal parameters
\$*	Alle parameters die zijn opgegeven
\$@	Alle parameters als gescheiden woorden

14.6 Het case-statement

Het *case-statement*, maakt gebruik van het voorkomen van bepaalde *cases* die in het script voorkomen. Er is van te voren al gedacht over de mogelijk uitvoer en springt daar op in. Maak de file 'case.sh' aan met de onderstaande code erin. Maak de file executable en voer het uit.

```
1 #!/bin/bash
2 echo -n "$*"
3 case $2 in
4     "-")
5         echo 'expr $1 - $3'
6         ;;
7     "+")
8         echo 'expr $1 + $3'
9         ;;
10    "*")
11        echo 'expr $1 * $3'
12        ;;
13 esac
14 exit
```

Op de tweede regel staat de opdracht `echo` die de waarde van alle variables die als parameter zijn opgegeven aan het script weergeeft (`$*`). Dus als we vanuit de shell intikken `./case 2 + 3` dan is de `$*` gelijk aan `'2 + 3'`. De optie `'echo -n'` zorgt ervoor dat er nog niet op een nieuwe regel wordt begonnen.

De shell leest nu het case-statement. Het vergelijkt de tweede parameter (`$2`) met een `'-'` of `'+'`. Mochten ze niet overeen komen dan voert hij de code uit die bij de wildcard (`*`) staat. De wildcard staat voor alle mogelijk tekens. De werking van het scripot is dus een comandline rekenmachine die getallen optelt en aftrekt.

De structuur van het case statement:

```
1  case "variable" in
2      mogelijkheid1)
3          statement1
4          statement2
5          .....
6      mogelijkheid2)
7          statement3
8          statement4
9          .....
10 esac
```

14.6.1 Case voorbeeld

Het volgende voorbeeld drukt telkens de komende dag af. Sla het op in de file morgen.sh, maak het executable en run het:

```
1 |#!/bin/bash
2 |date=date
3 |  case "$date +%a" in
4 |    Sun) echo Sunday ;;
5 |    Mon) echo Tuesday ;;
6 |    Tue) echo Wednesday ;;
7 |    Wed) echo Thursday ;;
8 |    Thu) echo Friday ;;
9 |    Fri) echo Saturday ;;
10 |   Sat) echo Sunday ;;
11 |   *) echo "Help! Onmogelijk om hier te komen!" ;;
12 | esac
```

Het script maakt op dezelfde manier als het voorbeeld hiervoor gebruik van het case-statement. Alleen neemt het nu als voorwaarde het eerste woord van `date` als resultaat. Het commando `date` wordt aan de variabele `$date` toegekent. Normaal doet het commando `date` dit:

```
1 | paul@slackbak:~$ date
2 | Mon Dec 20 13:55:43 CET 2010
```

Maar `'$date +%a'` is nu dus `Mon`, omdat `$date` het hele resultaat weergeeft. Als er `'$date +%b'` had gestaan was het resultaat `Dec`. Dit klinkt nogal abstract maar met een paar oefeningen is het goed te begrijpen.

14.6.2 HELP, dit script werkt niet!

Heb je het script netjes overgenomen, maar je krijgt "Help! Onmogelijk om hier te komen!"? Dat kan best. Het huidige script gaat ervan uit dat je Engels als standaard taal gebruikt. Dit is standaard bij in ieder geval *Slackware*, maar mogelijk bij een andere distributie niet. Op mijn laptop, met Nederlandse instellingen, krijg ik netjes deze fout. Hoezo? `date` geeft iets anders terug:

```
1 | paul@paul-laptop:/tmp$ date
2 | wo dec 8 18:39:39 CET 2010
```

Wat valt op? Juist, er staat geen `Wed`, maar `wo`. Het case statement zal dus nooit uitgevoerd worden! Hoe kan je dit oplossen? In dit geval vrij simpel, door expliciet de taal op te geven. Voor engels kan je dit doen door als taal `C` te gebruiken. Dit kan op de volgende manier:

```
1 | paul@paul-laptop:/tmp$ LANG=C ./morgen.sh
2 | Thursday
```

Bij het maken van een *Shell* script moet je altijd rekening houden met de taal van de gebruiker en het operating system. Er kan nooit van uitgegaan worden dan een gebruiker de standaard instelling of taal gebruikt. Eigenlijk kan er dus nooit in een *shell* script gecontroleerd worden op taal specifieke uitingen.

14.7 Het while statement

Het *while-statement* voert een bepaalde handeling uit totdat er aan een voorwaarde wordt voldaan. Hier een voorbeeldje, het script simuleert een dobbelsteen waarvan je het aantal ogen moet raden:

```
1  #!/bin/bash
2  trap 'echo bedankt voor het spelen.' EXIT
3  magischnr=$((RANDOM%6+1))
4
5  while [ "$gok" != "$magischnr" ]; do
6      echo Geef een cijfer tussen de 1 en de 6;
7      read gok
8  done
9  echo "Het magische nummer was: $magischnr"
10 exit
```

Het **trap** commando vangt de afbraak van het script af. Als er tijdens het runnen van het script plots op ctrl+c wordt gedrukt voert de shell het commando na de **trap** uit. Maar ook als het script netjes wordt afgesloten wordt het commando uitgevoerd. Meestal bevat een **trap** het verwijderen van tijdelijke files en weergeven van een afsluitboodschap.

De structuur van het while statement:

```
1  while conditie
2  do
3      statement1
4      statement2
5      .....
6  done
```

14.8 Het until-Statement

Het *until-statement* voert iets uit tot dat aan de voorwaarde wordt voldaan. Het lijkt op het while statement, alleen doet het een handeling pas als er aan een voorwaarde wordt voldaan, terwijl het while-statement zo lang er aan de voorwaarde wordt voldaan iets doet.

De structuur van het until statement:

```

1  until conditie
2  do
3      statement1
4      statement2
5      .....
6  done

```

Voorbeeld:

```

1  until ["$1" = "" ]
2  do
3      statement1
4      statement2
5      .....
6  done

```

14.9 Het for-statement

Naast de while en Untill loop hebben we ook nog de for loop. De for loop wijkt iets af van de uit *C* of *JAVA* bekende for loop. Er is standaard een teller, en die loopt een string af.

```

1  #!/bin/bash
2  for i in `seq 1 10`;
3  do
4      echo $i
5  done

```

seq print een sequence of nummers, in dit geval van 1 tot 10. De for loop zal hierna van 1 tot 10 tellen. De *i* is een variable welke gebruikt wordt als teller.

De structuur van de for loop is als volgt:

```

1 for i in "$string";
2 do
3     statement1
4     statement2
5     .....
6 done

```

14.10 Functies

In bijna elke taal zijn er functies aanwezig. Functies zijn handig binnen gebruik van programmeertalen, om bijvoorbeeld de code overzichtelijk te houden, veel voorkomende taken uit te voeren, of voor recursie. Ook in de *shell* zijn functies aanwezig. In het geval van een functie roep je hem gewoon

aan, net als een programma. Maak een bestand genaamd `aanwezig.sh` en zet het volgende script erin, en voer het hierna uit:

```
1 |#!/bin/bash
2 |aanwezig() {
3 |    if [ -e test.txt ] ; then
4 |        echo "bestaat"
5 |    else
6 |        echo "bestaat niet"
7 |    fi
8 |}
9 |aanwezig
```

Dit script maakt eerst een functie genaamd `aanwezig`. In deze functie is een controle of het bestand `test.txt` bestaat. (Zie paragraaf 14.4.3 voor uitleg over `-e`). Wanneer hij niet bestaat print hij dit netjes, en zodra hij bestaat print hij dit ook. Als laatste is nog de functie aanroep zelf. Dit is net als een bestand aanroepen, gewoon de functie naam. Een voorbeeld uitvoer is dit:

```
1 |paul@slackje:~$ ./aanwezig.sh
2 |bestaat niet
3 |paul@slackje:~$ touch test.txt
4 |paul@slackje:~$ ./aanwezig.sh
5 |bestaat
6 |paul@slackje:~$
```

In het eerste geval bestaat het bestand `test.txt` niet. Hierna wordt het bestand aangemaakt met het commando `touch`. Wanneer dit aangemaakt is, wordt het script nogmaals uitgevoerd. En nu bestaat hij!

14.10.1 Parameters

Het bovenstaande script is nog niet erg dynamische. Het zal altijd controleren voor één bepaald bestand. Wat je dus eigenlijk wilt is dat je dynamische kan opgeven om welk bestand dit gaat. Gelukkig is dit mogelijk, en wel op de volgende manier. Sla het volgende voorbeeld op als `aanwezig2.sh`:

```
1 |#!/bin/bash
2 |aanwezig() {
3 |    if [ -e $1 ] ; then
4 |        echo "$1 bestaat"
5 |    else
6 |        echo "$1 bestaat niet"
7 |    fi
8 |}
9 |aanwezig test.txt
10|aanwezig aanwezig2.sh
```

Dit script zal als voorbeeld dit als resultaat geven:

```

1 | paul@slackje:~$ ./aanwezig2.sh
2 | test.txt bestaat niet
3 | aanwezig2.sh bestaat
4 | paul@slackje:~$ touch test.txt
5 | paul@slackje:~$ ./aanwezig2.sh
6 | test.txt bestaat
7 | aanwezig2.sh bestaat

```

Het bovenstaande script is eigenlijk maar heel weinig veranderd. In plaats van `test.txt` staat er nu `$1`. De `$1` staat voor de eerste parameter aan die **functie** (Heeft het script ook als script parameters, dan zijn die niet zichtbaar voor de functie. Wil je de bestandsnaam toch zichtbaar maken, zal die apart op moeten geven worden als parameter). Dezelfde opties als voor de shell parameters uit paragraaf 14.5 gelden hier ook. Naast dat we afdrukken of het bestand dan daardwerkelijk bestaat, drukken we nu ook de bestandsnaam mee af. Dit zodat we kunnen zien wat de parameter eigenlijk was. De functie aanroep wordt nu ook aangepast. Als parameter moet er nu de bestandsnaam voor de controle worden opgegeven. Normaal gesproken zouden we ook nog moeten controleren of de parameter niet leeg is. Dit hebben we echter in dit voorbeeld weggelaten.

14.10.2 Shell parameters en functie parameters

Om het eerdere script nu nog iets verder aan te passen kunnen we ook nog de parameter dynamische maken door hem op te laten geven als shell parameter. Sla onderstaande code op als `aanwezig3.sh`.

```

1 | #!/bin/bash
2 | aanwezig() {
3 |     if [ -e $1 ] ; then
4 |         echo "$1 bestaat"
5 |     else
6 |         echo "$1 bestaat niet"
7 |     fi
8 | }
9 | aanwezig $1

```

Wat je hier ziet is dat de eerste Shell parameter direct doorgegeven wordt als parameter voor de functie, en daar wordt gebruikt. `$0` is de bestandsnaam van het Shell script, `$1` de eerste parameter. Zie ook paragraaf 14.5. Voorbeeld uitvoer:

```

1 | paul@slackje:~$ ./aanwezig3.sh
2 | bestaat
3 | paul@slackje:~$ ./aanwezig3.sh test.txt
4 | test.txt bestaat niet
5 | paul@slackje:~$ touch test.txt
6 | paul@slackje:~$ ./aanwezig3.sh test.txt
7 | test.txt bestaat

```

Zoals je ziet mist de controle op een lege parameter, en geeft hij dus dat een lege file aan als bestaat. Hierna wordt netjes gecontroleerd of test.txt aanwezig is.

14.11 Arrays

Met shell scripting kan er ook gebruik gemaakt worden van arrays. Alhoewel deze datasets hetzelfde zijn als in de meeste andere talen, is het syntactisch wel een vreemde eend. Er zullen daarom wat voorbeelden gegeven over het gebruik van arrays:

14.11.1 Assignment

Array assignment. De quotes dienen om aardbei ijs als een element te specificeren.

```
1| arr=( appel peer banaan "aardbei ijs" )
2| # Dit geeft dus de volgende array:
3| arr[0] = appel
4| arr[1] = peer
5| arr[2] = banaan
6| arr[3] = aardbei ijs
```

Ook kan de array assignment op specifieke locaties plaatsvinden:

```
1| arr=( [0]=appel [11]="aardbei ijs" )
```

14.11.2 Benaderen

Print het tweede element:

```
1| echo ${arr[1]}
```

Print de complete array:

```
1| echo ${arr[@]}
```

Print de array size:

```
1| echo ${#arr[@]}
```

Voeg een element toe aan de eerste vrije locatie

```
1| arr[${#arr[@]}]="nieuw"
```

Print de size van het tweede array element

```
1| echo ${#arr[1]} |
```

Print 2 elementen na id 10

```
1| echo ${arr[@]:10:2} |
```

Er is nog veel meer mogelijk met arrays, maar de meest gebruikte vormen zijn nu gegeven. Mocht er meer interesse zijn in de mogelijkheden kan er naar de *Advanced Bash Scripting Guide*[6] gekeken worden.

14.12 En nu?

Nu je de basis van shell scripting hebt gehad, kan je als het goed is het van SSH uit het vorige hoofdstuk compleet uit leggen hoe het werkt. Probeer zelf te bedenken hoe dit script werkt en wanneer je het niet snapt, lees de theorie nogmaals eens door en vraag je af wat je niet snapt.

14.13 Commando gebruik

Met alleen structuren kom je natuurlijk nergens. In veel programmeer talen zijn standaard functies aanwezig welke bepaalde taken kunnen doen. Met Shell scripting zal je hiervoor vaak moeten terugvallen op de standaard aanwezige commando's. Netals in de Shell zelf kan je in een Shell script standaard gebruik maken van de aanwezige commando's in de Shell. Voordat je deze gaat gebruiken moet je wel goed nadenken over wat voor consequenties dit heeft en hoe je dit kan opvangen.

We kunnen hier niet even alle commando's welke aanwezig zijn standaard in linux gaan uitleggen. Op een standaard Slackware install zijn dit er ruim 2600, en dit document is al lang genoeg. De belangrijkste commando's kan je lezen in hoofdstuk 12. Elk commando aanwezig in je distributie kan je in principe gebruiken in je Shell script. Hou er wel rekening mee dat je bijvoorbeeld in een *cron* niet vraagt om user input.

14.14 Alternatieve scripting mogelijkheden

Scripting is iets wat niet alleen mogelijk is door gebruik te maken van een van de *shells* op het systeem, maar dit kan ook gedaan worden met behulp van andere scripting talen.

Een veel gebruikte scripting-, en programmeer taal is `perl`[26][3]. De afkorting *perl* staat voor *Practical Extraction and Report Language*, een naam

die zeker recht wordt aangedaan.

`perl` is een scripting taal die erg gemakkelijk gebruikt kan worden om files (of streams) te analyseren. Dit komt omdat `perl` een enorme ondersteuning voor *reguliere expressies* heeft.

Eem een klein idee te geven over de syntax van een `perl` programma zal hieronder een klein scriptje te vinden zijn wat gebruikers uit het bestand `/etc/passwd` haald wanneer het `uid` van de betreffende gebruiker tussen de 0 en de 999 ligt. Ook zal de gebruiker zijn wachtwoord in `/etc/shadow` moeten hebben staan, of zal de login geblokkeerd moeten zijn. Het zal, wanneer de user gebruik maakt van de `sh` shell, deze omzetten naar `false`. Dit zal dan worden geprint:

```
1 |#!/usr/bin/perl
2 |open F, "/etc/passwd" or exit;
3 |foreach my $line (<F>) {
4 |    if($line =~ m/[a-z]+:[x|!]:\d{1,3}:/) {
5 |        $line =~ s/\sh/\false/;
6 |        print $line;
7 |    }
8 |}
```

Zoals te zien biedt het gebruik van `perl` een andere kijk op het toepassen van *scripting*. Het is ook zo dat er niet alleen gebruik gemaakt kan worden van `perl`, ook `python` is een taaltje wat vaak voor dit soort doeleinden wordt gebruikt.

Bijlage A

Slackware installeren op VirtualBox

Wanneer er geen computer vrij is om een nieuwe installatie op uit te voeren, is er de mogelijkheid om gebruik te maken van virtualisatie software. Een erg goed werkende, maar eenvoudige oplossing hiervoor is het gebruik van *VirtualBox*[17]. Je kan hier eenvoudig een virtuele computer creëren waar je vervolgens de installatie op uit kunt voeren.

Virtualbox kan eenvoudig geïnstalleerd worden. Op *Windows*, *Solaris* of *OS X* download je de installer van de virtualbox site. Onder *Linux* staat virtualbox meestal in de gebruikte package manager van de gebruikte distributie.

De installatie van *Slackware* is eigenlijk hetzelfde als normaal. Je mount de ISO van *Slackware* in de instellingen van *Virtualbox*. Hierna kan de VM gestart worden. Nu komt er een scherm krijgen met hierin de monitor over de VM. Volg hier gewoon alle normale installatie stappen uit hoofdstuk 3.

Zodra je klaar bent met de installatie moet je zelf de installatie CD unmounten via de instellingen. Wanneer je dit niet doet zal je iedere keer opnieuw booten in de installatie CD.

A.1 Problemen met netwerk/SSH

Virtualbox maakt standaard gebruik van NAT. Je kan dus niet zomaar bij een virtuele machine komen. Het kan echter wenselijk zijn om bepaalde services van de VM te ontsluiten, zodat je bijvoorbeeld naar de virtuele machine kan SSH'en. Dit is heel simpel te bereiken door op de *host* het volgende uit te voeren:

```
1 | kevin@iusaaset:~$ VBoxManage modifyvm "Slackware-current-i686"
   |   --natpf1 "guestssh,tcp,127.0.0.1,2022,,22"
```

Slackware-current-i686 is natuurlijk de naam van de VM. Verder staat hier dat je onder het label *guestssh*, met het TCP protocol verkeer gericht op 127.0.0.1:2022 wil forwarden naar guest:22.

Voer dit uit als de VM uit staat, en na het aanzetten van de VM is hij beschikbaar.

Bijlage B

BIOS

De BIOS is een belangrijk onderdeel van je computer. BIOS staat voor *Basic Input Output System*. Zonder BIOS zal je PC nooit werken. Wanneer je je PC opstart is het eerste wat je ziet je BIOS, nog voordat het operating system gestart is. De BIOS zal je basis hardware controleren en regelt alle communicatie tussen het operating system en de hardware.

In elke BIOS zijn instellingen aan te passen, en vaak heb je deze nodig om bijvoorbeeld Linux te kunnen installeren.

B.1 Boot sequence

Om Linux te kunnen installeren moet er vaak vanaf een ander medium als de aanwezige harde schijf opgestart worden. In veel gevallen is dit echter standaard uitgeschakeld en moet je de BIOS instellingen aanpassen naar je eigen wens zodat je van bijvoorbeeld CD of USB stick kan booten. Om dit te doen moet je opstarten in de BIOS. Vaak doe je dit door een toets zoals F2 of del in te drukken, welke dit is hangt af van de gebruikte hardware en wordt vaak weergegeven bij het opstarten.

Zodra je in de BIOS bent zal je op zoek moeten gaan naar de plek waar je de boot sequence aan kan passen. Vaak is dit een apart tabblad dat Boot Sequence heet. In dit tabblad zal je de sequence zo moeten aanpassen dat jou installatie medium voor de harde schijf staat. Zodra je dit gedaan hebt kan je het opslaan en opnieuw opstarten. Als het goed is start je PC nu op vanaf jou installatie medium.

Bijlage C

Partities en fdisk

Een harddisk wordt meestal ingedeeld in partities. Voor de installatie van *Linux* moeten er minimaal twee partities worden gemaakt. Één partitie voor het algemene *Linux filesystem* en één partitie voor swap. Windows partitioneert verplicht in een primaire en een extended partitie. Binnen de extended partitie kunnen vervolgens weer enkele logical partities aangemaakt worden. *Linux* ondersteunt echter meerdere primaire partities.

C.1 Fdisk

Om je SATA schijf te partitioneren moet je het volgende commando gebruiken:

```
1| fdisk /dev/sda
```

Dit commando zal de eerste aanwezige schijf nemen. Wil je een andere schijf? Verander dan de sda in bijvoorbeeld sdb. Als je het niet zeker weet, kan je via het command

```
1| dmesg | less
```

dit vinden.

Type nu eerst m in om te zien welke opties fdisk allemaal heeft. Je krijgt onderstaand scherm:

```
1| root@slackje:~# fdisk /dev/sda
2|
3| Command (m for help): m
4| Command action
5|   a   toggle a bootable flag
6|   b   edit bsd disklabel
7|   c   toggle the dos compatibility flag
8|   d   delete a partition
```

```

9 | l list known partition types
10 | m print this menu
11 | n add a new partition
12 | o create a new empty DOS partition table
13 | p print the partition table
14 | q quit without saving changes
15 | s create a new empty Sun disklabel
16 | t change a partition's system id
17 | u change display/entry units
18 | v verify the partition table
19 | w write table to disk and exit
20 | x extra functionality (experts only)
21 |
22 | Command (m for help):

```

Kies vervolgens 'p' om de huidige partitie instellingen te tonen.

```

1 | Command (m for help): p
2 |
3 | Disk /dev/sda: 26.8 GB, 26843545600 bytes
4 | 255 heads, 63 sectors/track, 3263 cylinders, total 52428800
   | sectors
5 | Units = sectors of 1 * 512 = 512 bytes
6 | Sector size (logical/physical): 512 bytes / 512 bytes
7 | I/O size (minimum/optimal): 512 bytes / 512 bytes
8 | Disk identifier: 0x028f6a13
9 |
10 |
11 | Device Boot      Start         End      Blocks   Id  System
12 | /dev/sda1  *            63      10506509   5253223+  83  Linux
13 | /dev/sda2                10506510   41977844  15735667+  83  Linux
14 | /dev/sda3                41977845   52420094   5221125   82  Linux
   | swap

```

Hier zijn al drie partities aanwezig. Deze handleiding gaat er van uit dat er geen partities aanwezig zijn, en deze partities hierboven zullen dan ook niet meer genoemd worden hieronder. Wanneer dit wel het geval is moet het eerstvolgende volg nummer gekozen worden. In dit geval is dat 4. Deze waarde moet onthouden worden. Kies vervolgens 'n' om een partitie toe te voegen.

```

1 | Command (m for help): n
2 | Command action
3 |   e extended
4 |   p primary partition (1-4)
5 | p
6 | Partition number (1-4, default 1): 1
7 | First sector (2048-52428799, default 2048): 2048
8 | Last sector, +sectors or +size{K,M,G} (2048-52428799, default
   | 52428799): +10240M

```

Na 'n' is achtereenvolgens gekozen voor 'p', '1', '1'¹ en '+10240M'. De 'p' staat voor primary partitie en de '1' voor het nummer. Vervolgens wordt om het cylinder nummer gevraagd. De waarde '1' geeft aan dat het begin van de harddisk gekozen wordt. Vervolgens dient de eind-cylinder van de partitie gegeven te worden. De waarde '+10240M' staat voor 10 gigabte. Nu de hoofdpartitie is aangemaakt kan de swap partitie gemaakt worden. Hiervoor kan dezelfde procedure gebruikt worden, namelijk: 'n', 'p', '2', 'enter', '+128M' (Of meer²). Nu is de tweede partitie aangemaakt met een grote die jij zelf gekozen hebt. Kies vervolgens 'p' om de partitie tabel te weergeven.

```

1| Command (m for help): n
2| Command action
3|   e   extended
4|   p   primary partition (1-4)
5| p
6| Partition number (1-4, default 2):
7| Using default value 2
8| First sector (20973568-52428799, default 20973568):
9| Using default value 20973568
10| Last sector, +sectors or +size{K,M,G} (20973568-52428799,
    |   default 52428799): +128M
11|
12| Command (m for help): p
13|
14| Disk /dev/sda: 26.8 GB, 26843545600 bytes
15| 255 heads, 63 sectors/track, 3263 cylinders, total 52428800
    |   sectors
16| Units = sectors of 1 * 512 = 512 bytes
17| Sector size (logical/physical): 512 bytes / 512 bytes
18| I/O size (minimum/optimal): 512 bytes / 512 bytes
19| Disk identifier: 0x028f6a13
20|
21|   Device Boot      Start         End      Blocks   Id  System
22| /dev/sda1                2048     20973567     10485760    83  Linux
23| /dev/sda2            20973568     21235711      131072    83  Linux

```

De partitie tabel geeft 2 partities weer. Beide hebben id 83 (Linux native). Dit is het standaard Linux type. Voor de tweede partitie moet dit veranderd worden in id 82 (Linux swap). Kies 't' voor het veranderen van het type, en vervolgens het partitie nummer '2'. Nu kan het id gegeven worden (82), of via de letter 'L' kan er een lijst van typen worden opgevraagd. Na het veranderen van het type kan via de optie 'p' een nieuw partitie overzicht gevraagd worden. Dit zal er dan zo uit zien:

```

1| Command (m for help): t

```

¹In het voorbeeld is gekozen voor 2048, omdat deze harde schijf in gebruik is. Echter, bij een nieuwe installatie kies je voor 0.

²Het best kan je kiezen voor een SWAP partitie van 2 maal je RAM geheugen. Als je dus 2GB RAM hebt kan je het beste 4GB swap nemen.

```

2 | Partition number (1-4): 2
3 | Hex code (type L to list codes): 82
4 | Changed system type of partition 2 to 82 (Linux swap)
5 |
6 | Command (m for help): p
7 |
8 | Disk /dev/sda: 26.8 GB, 26843545600 bytes
9 | 255 heads, 63 sectors/track, 3263 cylinders, total 52428800
   | sectors
10 | Units = sectors of 1 * 512 = 512 bytes
11 | Sector size (logical/physical): 512 bytes / 512 bytes
12 | I/O size (minimum/optimal): 512 bytes / 512 bytes
13 | Disk identifier: 0x028f6a13
14 |
15 |   Device Boot      Start         End      Blocks   Id  System
16 | /dev/sda1                2048     20973567    10485760    83  Linux
17 | /dev/sda2            20973568     21235711     131072    82  Linux
   | swap

```

Als laatste moet de optie 'a' gekozen worden om partitie '1' bootable te maken. Deze optie is van belang voor het installeren van een bootloader zoals LILO en voor het maken van een multi-boot systeem met bijvoorbeeld Windows. En het eindresultaat is dan zo:

```

1 | Command (m for help): a
2 | Partition number (1-4): 1
3 |
4 | Command (m for help): p
5 |
6 | Disk /dev/sda: 26.8 GB, 26843545600 bytes
7 | 255 heads, 63 sectors/track, 3263 cylinders, total 52428800
   | sectors
8 | Units = sectors of 1 * 512 = 512 bytes
9 | Sector size (logical/physical): 512 bytes / 512 bytes
10 | I/O size (minimum/optimal): 512 bytes / 512 bytes
11 | Disk identifier: 0x028f6a13
12 |
13 |   Device Boot      Start         End      Blocks   Id  System
14 | /dev/sda1  *            2048     20973567    10485760    83  Linux
15 | /dev/sda2            20973568     21235711     131072    82  Linux
   | swap

```

Kies 'w' om de partitie instellingen op te slaan en fdisk te verlaten.

Als je zelf een filesystem wilt maken op een partitie kan dit bijvoorbeeld met `mkfs`. Zie man `mkfs` voor meer informatie.

C.2 Swap

Het kan in sommige gevallen gebeuren dat de swap partitie niet in */etc/fstab* staat. Om te controleren of je *swap* hierin staat moet je het volgende doen:

```
1| pico /etc/fstab
2| [...]
3| /dev/sda3          swap      swap    sw      0      0
```

Wanneer er geen regel instaat met *swap* kan je de bovenstaande regel toevoegen. Hierbij is */dev/sda3* natuurlijk de *swap partitie* die je zelf hebt gemaakt. Na het handmatig inschakelen van de *swap partitie* via *swapon*, of na een herstart zal *deswap* wel werken. Dit kan met *free* worden gecontroleerd:

```
1| paul@slackbak:~$ free -m
2|      total        used          free      shared    buffers
3| Mem:      1001         78           923          0         19
4|    31
5| -/+ buffers/cache:      26           974
6| Swap:      3067          0          3067
```

In de kolom “free” van de regel *swap* zal bij een werkende *swap* een getal groter dan nul staan.

Bijlage D

Boot manager

Om een systeem te kunnen starten is er de mogelijkheid om gebruik te maken van zogenaamde *boot managers*. Dit zijn programma's die het mogelijk maken om op een generieke manier een besturingssysteem de mogelijkheid te geven om te starten.

D.1 MBR

De afkorting *MBR* staat voor *master boot record*, de eerste sector van een gepartitioneerde schijf. Het is een sector van 512 bytes, waardoor het een minimale beschikbaarheid heeft voor opslag. De data die hier op staat, kan een of meer van de volgende punten zijn.

1. Het bevatten van de partitie tabel van een schijf.
2. Het starten van een boot manager.
3. Een uniek device id opslaan.

D.2 Bootable partitie

Wanneer er geen gebruik wordt gemaakt van de *MBR* om de bootmanager onder te brengen, zal de *BIOS* van de computer kijken in de *bootsector* van de eerste harde schijf. Er zal dan een *boot manager* vanaf de partitie worden gestart. Met `fdisk` is te zien welke partitie *bootable* is, de *bootable* flag is eenvoudig met `fdisk` te wijzigen.

Wanneer met `fdisk` de partities worden gelist zal de partitie met "*" de partitie zijn die *bootable* is.

```

1 | root@slackje:~# fdisk /dev/sda
2 | [...]
3 | Command (m for help): p
4 | [...]
5 |   Device Boot Start      End          Blocks      Id System
6 | /dev/sda1 *    63           10506509    5253223+    83 Linux
7 | /dev/sda2      10506510    41977844    15735667+    83 Linux
8 | /dev/sda3      41977845    52420094    5221125      82 Linux swap

```

Wanneer de *bootpartitie* veranderd dient te worden, kan er weer gebruik worden gemaakt van **fdisk**. Dit is met de optie “a” erg makkelijk te regelen.

```

1 | root@slackje:~# fdisk /dev/sda
2 | Command (m for help): a
3 | Partition number (1-4): 2
4 | Command (m for help): p
5 | [...]
6 |   Device Boot Start      End          Blocks      Id System
7 | /dev/sda1 *    63           10506509    5253223+    83 Linux
8 | /dev/sda2 *    10506510    41977844    15735667+    83 Linux
9 | /dev/sda3      41977845    52420094    5221125      82 Linux swap

```

Omdat de optie “a” de boot status van een schijf alleen maar aan of uit zet hebben we hier een potentiële fout situatie. Een schijf mag namelijk maar een *bootable* partitie hebben. We zullen de andere dus nog even uit moeten zetten.

```

1 | root@slackje:~# fdisk /dev/sda
2 | Command (m for help): a
3 | Partition number (1-4): 1
4 | [...]
5 |   Device Boot Start      End          Blocks      Id System
6 | /dev/sda1      63           10506509    5253223+    83 Linux
7 | /dev/sda2 *    10506510    41977844    15735667+    83 Linux
8 | /dev/sda3      41977845    52420094    5221125      82 Linux swap

```

D.3 GRUB

In het geval van *GRUB*¹ gebeurt dit op de volgende manier.

De *BIOS* zal het primary boot device selecteren, en deze laden. In het geval van *GRUB* is dit de zogenaamde *GRUB stage 1*². Deze sectie kan vervolgens gelijk *GRUB stage 2* laden, of het kan uit de eerste 30 kilobytes van de harde schijf een tussenstap laden, *GRUB stage 1.5*. Deze stap kan gebruikt worden om extra *file system* drivers te laden zodat in *stage 2* een systeem gestart kan worden.

¹Dit gaat over *GRUB 1*.

²De *MBR* methode werkt vanaf hier identiek.

Wanneer *GRUB* zich in stage 2 bevindt krijgt de gebruiker ook de mogelijkheid om met een kleine *command line* eventueel een alternatieve boot optie in te voeren. Dit kan gebruikt worden om het systeem met andere waardes op te starten, of om juist een rescue procedure te starten.

Door de flexibele opzet is *GRUB* ook erg geschikt voor systemen waar het niet direct compatible mee is. Het is namelijk mogelijk voor *GRUB* een zogenaamde *chainload* kan uitvoeren. Het geeft dan als het ware het stokje door aan de volgende *boot manager* in rij. De *boot manager* die dan actief wordt heeft totaal geen weet heeft dat het vooraf is gegaan door *GRUB*. Dit biedt flexibiliteit om bijvoorbeeld een *Windows* systeem te starten vanuit *GRUB*.

D.3.1 GRUB installeren

Het installeren van *GRUB* is erg gemakkelijk. Er bestaat een *GRUB package* die in de *extra* tak van de *repository* te vinden is. Wanneer de package is geïnstalleerd kan deze worden geconfigureerd.

```
1| root@slackbak:/# wget ftp://ftp.fu-berlin.de/unix/linux/mirrors/ |
   | slackware/slackware-13.37/extra/grub/grub-0.97-i486-9.tgz
2| root@slackbak:/# installpkg grub-0.97-i486-9.tgz
3| root@slackbak:/# grubconfig
```

Nu kan er voor “simple” worden gekozen, gevolgd door de “standard” *framebuffer* optie. De volgende stap is het opgeven van de partitie met de “/boot”. Geef hier bijvoorbeeld */dev/sda1* op. De laatste optie is de keuze tussen een installatie in de *MBR* of de partitie van de schijf. In het laatste geval zal de partitie weer bootable verklaard moeten zijn.

D.3.2 GRUB boot wijzigen

GRUB biedt gebruikers de mogelijkheid om met gewijzigde boot parameters te starten. Ook kan er gekozen worden om tijdelijk van een andere partitie te starten. Dit kan door in het *GRUB* menu op de “e” toets te drukken. Hierdoor kan de huidige geselecteerde entry gemodificeerd worden.

Om van een andere partitie te booten kan bijvoorbeeld worden gekozen om de optie *root* te veranderen.

```
1| root (hd1,1)
```

Wanneer dit als root optie word gegeven betekend dit dat er geprobeerd zal worden om vanaf de tweede schijf en de tweede partitie te starten.

Een andere handigheid is om de *init* parameter van de kernel te overschrijven. Een voorbeeld is het toevoegen van het volgende.

```
1| init=/bin/bash
```

Dit houdt in dat er gestart zal worden met de *bash* shell als *init* proces. Dit truukje wordt uitgebreider uitgelegd in bijlage E.

Wanneer alle opties naar wens zijn aangepast kan het systeem worden gestart met de “b” toets.

Wanneer deze wijzigingen permanent dienen te zijn kunnen de *GRUB* opties in het volgende bestand permanent worden aangepast.

```
1| /boot/grub/menu.lst
```

D.4 LILO

Op oudere systemen, of op standaard *Slackware* omgevingen komt het ook nog voor dat er *LILO* wordt gebruikt. Dit is de voorganger van *GRUB*, en staat voor *LI*nux *LO*ader, een boot manager om *Linux* te kunnen starten.

D.4.1 Installatie

Het *Slackware* commando *liloconfig* is een menu gebaseerd configuratie programma waarmee *LILO* eenvoudig geconfigureerd kan worden.

Om *LILO* te installeren kan tijdens het draaien van *liloconfig* gekozen worden voor een “simple” install, gevolgd door een standaard *framebuffer* keuze. De optionele *kernel* parameters kunnen hoogstwaarschijnlijk leeg worden gelaten, tenzij je een goede reden hebt om er een toe te voegen. De keuze voor *UTF-8* haalt momenteel niet meer uit, software is op nieuwe systemen hier wel op voorbereid. Bij de laatste keuze moet er gekozen worden voor een locatie om *LILO* te installeren.

In de bovenstaande uitleg wordt er vanuit gegaan dat er geïnstalleerd wordt in de *bootable* partitie. De *MBR* is ook zeker een mogelijke optie, kies dan natuurlijk voor die optie.

D.4.2 Aanpassen LILO configuratie

De configuratie van *LILO* vindt plaats in */etc/lilo.conf*. De belangrijkste is de regel met *boot* erin.

```
1| boot=/dev/sda
```

Hier staat de locatie van *LILO*. Momenteel is dit de *bootsector* van de eerste harde schijf, dus niet de *root* partitie. Dit werkt alleen wanneer ook de *boot manager* in de *MBR* te vinden is. In het eerder beschreven scenario zal dit dus moeten worden aangepast.

```
1| boot=/dev/sda1
```

Bijlage E

Rescue

Ook *Linux* is een systeem waar door verschillende fouten er een slecht of totaal niet werkend systeem kan optreden. Door de structurele opzet van veiligheid zul je echter wel fysieke toegang moeten hebben voordat je een poging tot herstel kan wagen. Wanneer je fysiek met het systeem aan de slag kan, zijn er wel verschillende mogelijkheden om het systeem te redden. Is er geen mogelijkheid om toegang te hebben tot het fysieke systeem, dan ben snel verloren.

E.1 Nieuwe harddisk

Wanneer er een nieuwe harddisk in het systeem wordt gedaan, zou het kunnen zijn dat het systeem niet meer opstart. Wanneer dit het geval is, dien je na te denken over wat je nu precies hebt toegevoegd, en in hoeverre dit verschilt van de originele situatie.

Wanneer de kernel sata devices gaat selecteren, zullen alle beschikbare devices worden genummerd. Dit resulteert in de volgende tabel. De volgorde is afhankelijk van de volgorde waarop de schijven worden aangesproken.

Device node	Betekenis
/dev/sda	Disk 1
/dev/sdb	Disk 2
/dev/sdX	Disk X

Om het overzicht compleet te houden zullen we ook een kleine enumeratie geven van een systeem met *IDE* schijven. Vroeger werden deze */dev/hdX* genoemd, maar sinds de Linux kernel IDE schijven ook door *libata* laat afhandelen hebben deze ook een */dev/sdX* naam gekregen. In de praktijk levert het de volgende situatie op.

Device node	Betekenis
/dev/sda	Primary master
/dev/sdb	Primary slave
/dev/sdc	Secondary master
/dev/sdd	Secondary slave

Nu er extra device(s) zijn toegevoegd, zou het zo kunnen zijn dat de enumeratie anders is. Het zou kunnen zijn dat het systeem niet meer boot, met een *GRUB ERROR 11*. Dit betekent dat er een filesystem is gemount, maar er geen kernel kan worden gevonden. Dit komt meestal omdat de device volgorde is gewijzigd.

Om dit te herstellen zullen we *GRUB* moeten laten starten met een aangepaste boot optie. zie D.3.2 *GRUB boot wijzigen* om te kijken hoe dit moet. Zorg dat je nieuwe boot device het correcte oude *boot device* is, en pas het daarna permanent aan. Het systeem zal nu opstarten.

Wanneer er nu nog wordt geklaagd over het niet kunnen vinden van het *root filesystem* dien je de file */etc/fstab* te wijzigen naar de nieuwe situatie.

Bij deze paragraaf is wel een kanttekening te maken. *Slackware* is een erg traditionele distributie, welke vaak de kat uit de boom kijkt bij nieuwigheden, soms zelfs jarenlang. Practisch heeft dit gevolgd dat er vaak relatief oude, maar bewezen technieken gebruikt worden.

Dit houdt echter in dat nieuwe trends, zoals het gebruik van *UUID's* iets is wat standaard ongebruikt is. Men kan het zelf configureren, maar het zal standaard niet in gebruik zijn. Nu creëert het gebruik van *UUID's* een scenario waar het eerder beschreven systeem zich niet kan voordoen, omdat ieder device een gegarandeerd uniek ID krijgt. Voor de werking hiervan willen we jullie naar *Universally Unique Identifier*[29] verwijzen.

E.2 Root password gecompromitteerd

Opnieuw installeren. Je systeem is niet meer veilig. Breng alle gerelateerde mensen op de hoogte van je probleem. Ga ervan uit dat al je data is gekopieerd en dat alle NAW gegevens zijn verkocht. Zoek uit hoe dit zich heeft kunnen voordoen, en neem voorzorgsmaatregelen om dit te voorkomen.

Geen enkel commando hoeft meer correcte informatie terug te geven. Een hacker kan in theorie ieder commando aanpassen zodat het informatie naar zijn wensen teruggeeft. Dit kan gaan van een simpele aanpassing naar *w* of *who*, maar ook naar commando's als *halt* en *reboot*.

E.3 Root password vergeten

Dit is iets wat niet slim is, maar het is geen ramp. Om het systeem te kunnen redden zul je het wel moeten herstarten. Gebruik de sectie D.3.2 *GRUB boot wijzigen* om het systeem te starten met een gewijzigde parameter. Wanneer de parameter niet bestaat, voeg hem dan toe aan de *kernel* regel.

```
1| init=/bin/bash
```

Na enkele seconden zie je nu de bekende *BASH* shell. Gebruik deze om handmatig, of met *passwd* het root wachtwoord te laten wijzigen. Start nu het programma *sync*, wacht 3 seconden, en start dan opnieuw op.

E.4 Windows geïnstalleerd - Linux weg

Onze vrienden van Microsoft zijn zo sociaal om bewust een jarenlange bug te laten bestaan. Bij een installatie van Windows word er niet gekeken of er eventueel al een besturingssysteem te vinden is, en zo ja; of deze gebruik maakt van de *MBR* van de harde schijf. Hierdoor zal de bootloader worden overschreven met een speciale versie van Windows, welke dus je Linux installatie negeert. Dit repareren vereist het starten van een Linux cd, waarna je grub opnieuw laat installeren. Zie hiervoor D.3.1 *grub installeren* voor *Slackware* specifieke instructies. De meeste systemen voldoen met het draaien van het volgende commando:

```
1| grub-install /dev/sda
```

Hier geldt natuurlijk weer dat */dev/sda* het goede device moet zijn. Beantwoord eventuele vragen, en geniet van een werkende installer. Controleer wel even over er een Windows entry te vinden is, of voeg deze toe. Een voorbeeld van zo'n *chainloader* is als volgt. We gaan uit van disk 2, partitie 3.

```
1| title Windows savedefault 0,0
2| root (hd1,2)
3| savedefault
4| makeactive
5| chainloader +1
```

Bijlage F

Slackware Packages

Package management betekent dat er kant en klare software pakketten gedownload kunnen worden. De gebruiker zal dus geen compilatie van de source files hoeven te doen. Dit maakt het installeren van software een stuk gemakkelijker en sneller. Slackware biedt een minimale package manager aan, maar toch is het erg handig om te weten hoe dit werkt. We zullen dit hier uitleggen.

F.1 pkgtool

Dit is een frontend voor de verschillende losse onderdelen van de package manager. Het is via een grafische interface te bedienen. Het kan onder andere gebruikt worden om geïnstalleerde software te bekijken, nieuwe software te installeren of oude software te verwijderen. Voor specifiekere toepassingen is het echter gemakkelijker om terug te grijpen op de programma's die onder de motorkap worden gebruikt.

F.2 installpkg

Wanneer je een package download kan je met dit programma de package installeren.

```
1|root@slackbak:/tmp# installpkg grub-0.97-i486-9.txz |
```

F.3 upgradepkg

Dit kan gebruikt worden om packages te updaten. Er zijn twee manieren om dit commando te gebruiken. Het upgraden van een package met dezelfde naam.

```
1 root@slackbak:/tmp# upgradepkg grub-0.97-i486-9.txz
2
3 +-----+
4 | Upgrading grub-0.95-i486-2 package using ./grub-0.97-i486-9.
   | txz
5 +-----+
6 [...]
```

Het upgraden van een package die hetzelfde aanbied, maar een nieuwe naam heeft gekregen kan ook. De syntax is dan oude naam%nieuwe naam.

```
1 root@slackbak:/tmp# upgradepkg oudenaam-grub-0.95-i486-2.txz%
   grub-0.97-i486-9.txz
2
3 +-----+
4 | Upgrading oudenaam-grub-0.95-i486-2 package using ./grub-0.97-
   | i486-9.txz
5 +-----+
6 [...]
```

F.4 removepkg

Hiermee kunnen packages worden verwijderd.

```
1 root@slackbak:/tmp# removepkg grub-0.97-i486-9
2 Removing package /var/log/packages/grub-0.97-i486-9...
3 Removing files :
4 [...]
```

Bijlage G

Source installatie

Van veel populaire softwarepakketten zullen Slackware packages beschikbaar zijn. Deze packages hebben een aantal voordelen, waardoor ze gemakkelijk en gebruiksvriendelijk zijn in de praktijk. Voor meer details, zie bijlage F *Slackware Packages*.

G.1 Achtergrond

Wanneer een bepaald stuk software niet beschikbaar is als Slackware package betekent het niet dat het onmogelijk is dit pakket te gebruiken. Er is echter alleen geen gebruiksvriendelijke manier om de packages te installeren. Gebruikers dienen in dit geval zelf een zogenaamde *source install* te doen.

Omdat problemen tijdens een *source install* vrij cryptisch over kunnen komen, zullen we naast de uitleg van de installatie zelf nog het een en ander vertellen over de tools die worden gebruikt voor de installatie. Dit kan lezers van het document eventueel goed op weg helpen om eventuele problemen op te lossen.

G.1.1 Voordelen

Het grootste voordeel van een source installatie is de flexibiliteit. Je bent niet afhankelijk van de meegecompileerde opties die een packager voor jou logisch vindt. Wanneer er gebruik gemaakt wordt van repositories gebeurt het namelijk regelmatig dat er features missen in software die je eigenlijk toch wel wilt gebruiken. Deze features hebben vaak een *-enable* flag in het configure script.

Ook is software vaak nieuwer dan uit repositories. Je kunt immers zelf kiezen

welke versie je wilt installeren. Niemand zal je dus tegenhouden wanneer je een beta versie wilt gebruiken. Ook hoeft je niet te wachten totdat de packager tijd heeft gehad om de gebruikte package te updaten.

G.1.2 Nadelen

Er zijn echter wel een aantal erg belangrijke nadelen. De grootste is de extra tijd die je kwijt bent aan onderhoud. Wanneer je zelf software compileert, en je wilt je pakket upgraden zal je de source moeten downloaden, opnieuw *configure* moeten runnen, daarna eventuele dependencies resolvable, om vervolgens weer een complete compilatie van het pakket te doen. Dit is een tijdrovende business.

Een ander groot nadeel is het gebrek aan updates. Door het bovenstaande nadeel zullen source installs vaak achterlopen op stable releases van het software pakket. Wanneer er (grote) beveiligingslekken te vinden zijn, en je dit niet direct door hebt ben je langer kwetsbaar voor aanvallen. Je zult dus actiever je software bij moeten houden.

G.2 Hoe gaan we te werk

Een source installatie zelf is vrij simpel, het is puur de compilatie van de broncode van een bepaald project. Deze compilatie heeft echter verschillende afhankelijkheden van het systeem waar het op draait, zodat er vaak erg unieke situaties ontstaan. Deze verschillen tussen systemen zijn vaak in de loop der tijd ontstaan, en hebben meestal te maken met de locatie van de zogenaamde *dependencies*, afhankelijkheden voor de compiler en de linker. Een andere oorzaak zijn verschillen in systemcalls tussen verschillende besturingssystemen.

In dit voorbeeld zullen we het programma *netcat* installeren. De website hiervan is hier te vinden: <http://netcat.sourceforge.net>.

G.2.1 Source download

Om te beginnen hebben we de sourcecode van het programma nodig. Download versie 0.7.1. Dit is op het moment van schrijven de meest recente versie.

```
1 kevin@slackbak:~$ wget http://garr.dl.sourceforge.net/
   sourceforge/netcat/netcat-0.7.1.tar.bz2
2 [...]
3 2010-12-13 11:32:45 (2.31 MB/s) - "netcat-0.7.1.tar.bz2" saved
   [325687/325687]
```

Het is aan te raden bij belangrijke software de integriteit van het programma te controleren. Zo weet je dat de download niet corrupt is geraakt, of dat deze eventueel door malafide derde partijen is aangepast. Dit kan als volgt.

```
1| kevin@slackbak:~$ wget http://netcat.sourceforge.net/signatures/ |
   | md5sums.txt
2| kevin@slackbak:~$ md5sum -c md5sums.txt netcat-0.7.1.tar.bz2 2>
   | /dev/null | grep netcat-0.7.1.tar.bz2
3| netcat-0.7.1.tar.bz2: OK
```

Nu we weten dat het archief correct is gedownload kunnen we het gaan uitpakken.

```
1| kevin@slackbak:~$ tar -xjf netcat-0.7.1.tar.bz2
```

Wanneer er gebruik wordt gemaakt van een *gzip* in plaats van een *bzip2* zal de “j” vervangen moeten worden met een “z”.

Als de *signature* niet correct was, is het verstand het archief opnieuw te downloaden. Blijkt bij deze tweede controle weer een foute *MD5* signature, is er (Als die optie aangeboden wordt) nog een mogelijkheid tot downloaden via een andere mirror. Het kan echter wel slim zijn de maker te informeren over een mogelijke aangepast archief, door derde personen.

G.2.2 configure

Om alle systeem afhankelijkheden te onderzoeken kan het *configure* script worden gedraaid. Dit zal de zogenaamde *Makefiles* genereren welke gebruikt kunnen worden om de software te compileren. *Configure* zorgt er ook voor dat de platform specifieke compileer instructies worden toegepast.

Deze file is in bijna alle *GNU* source trees te vinden, het is ook deel van de richtlijnen voor *GNU* software. Dit komt omdat de *GNU Foundation* programmeurs aanraad om allemaal gebruik te maken van de *GNU Build System*[10]. Dit geeft programmeurs veel handige utilities om een source tree zo neer te zetten dat deze op een generieke manier te gebruiken is.

Om alle opties van het *configure* script te bekijken kan er om help informatie gevraagd worden.

```
1| kevin@slackbak:~$ cd netcat-0.7.1/
2| kevin@slackbak:~/netcat-0.7.1$ ./configure --help
```

Er zullen nu heel veel opties van het *configure* script worden geprint. Veel ervan zullen een goede default value hebben, maar een gebruiker zou default values aan kunnen willen passen. Eventueel kunnen er extra parameters op gegeven worden.

We zullen zorgen dat we netcat als gewone gebruiker kunnen installeren. Je hoeft dus geen root te zijn om software te kunnen installeren.

```
1| kevin@slackbak:~/netcat-0.7.1$ ./configure --prefix=$HOME/
   software/netcat
```

Nu zullen alle systeem checks worden uitgevoerd. Hierna zullen ook alle *makefiles* worden voorbereid om te installeren in de `software/netcat` directory van je home map.

G.2.3 make

Wanneer het script klaar is met uitvoeren en er geen fouten gemeld zijn, zullen alle makefiles klaargemaakt zijn, en kan er worden begonnen met het compileren van de software. Dit is een erg simpele stap.

```
1| kevin@slackbak:~/netcat-0.7.1$ make
```

Alle sourcecode zal nu worden gecompileerd zodat we het daadwerkelijke programma krijgen. Mocht de compilatie onverhoopt toch mislukken zal *Make* dit melden. Je kan dan de code eventueel *patchen* zodat deze wel compileert.

G.2.4 make install

Nu *Make* de code succesvol heeft gecompileerd is het mogelijk om de code te installeren. Ook dit is weer erg simpel omdat er een *install* target is gespecificeerd.

```
1| kevin@slackbak:~/netcat-0.7.1$ make install
```

Nu zal make de gecompileerde binaries installeren in de vooraf opgegeven locaties. In ons geval zal er dus een installatie te vinden moeten zijn in `~/software/netcat`.

```
1| kevin@slackbak:~$ file software/netcat/bin/netcat
2| software/netcat/bin/netcat: ELF 32-bit LSB executable, Intel
   80386, version 1 (SYSV), dynamically linked (uses shared libs
   ), not stripped
```

De binary is er inderdaad te vinden. Nu kunnen we het nog testen.

```
1| kevin@slackbak:~$ export PATH=$PATH:software/netcat/bin/
2| kevin@slackbak:~$ which netcat
3| /home/kevin/software/netcat/bin/netcat
4| kevin@slackbak:~$ netcat --help
5| GNU netcat 0.7.1, a rewrite of the famous networking tool.
```

Netcat is nu succesvol geïnstalleerd, in de home map van de gebruiker.

G.2.5 Source management

Om de geïnstalleerde software te kunnen deïnstalleren is de makkelijkste optie om de source directory te bewaren. Ook kan dit handig of nodig zijn wanneer er software word gecompileerd die een afhankelijkheid heeft op een eerder pakket.

Het is gebruikelijk om de sources op een vaste locatie te bewaren, en niet zomaar in de home map van de gebruiker zoals hierboven beschreven. In principe zijn de volgende conventies aan te raden.

1. Systeem brede sources.

1| `/usr/local/src` |

2. Gebruiker specifieke sources.

1| `$HOME/src/` |

Op deze manier staat alle software op een algemene locatie, en kan er wanneer nodig gemakkelijk een uninstall target gerund worden. Mocht je software hebben die afhankelijk is van een ander pakket, is het ook gemakkelijk hier naartoe te verwijzen.

Bijlage H

SSH

Van het *SSH* protocol bestaan meerdere versies. We zullen dit verhaal richten op versie twee hiervan. Dit doen we omdat versie twee de variant is die standaard wordt gebruikt in productie omgevingen, aangezien deze veiliger is.

De details over deze beslissing vallen buiten de scope van dit document, hiervoor verwijzen we naar de encryptievakken van de opleiding. Dit geldt voor deze complete sectie van de appendix. We zullen, wanneer relevant, wel verwijzen naar externe informatiebronnen.

H.1 Verbinding opzetten

Wanneer er een verbinding wordt opgezet zal er als eerste stap een *Diffie-Hellman*[14][9] key agreement plaatsvinden. Hieruit volgt een zogenaamde *shared session key*. Deze shared session key is een geheime sleutel welke bekend is bij de beide partijen, maar toch niet bekend kan zijn bij partijen die eventueel de communicatie afluisteren. Er is dus een geheime key te genereren over een onveilig medium.

De rest van de sessie verloopt via een *symmetrische versleuteling*[28]. Er is hiervoor de mogelijkheid om uit verschillende algoritmes te kiezen. Keuzes worden doorgegeven aan de client, welke er een zal uitkiezen voor de communicatie. De sleutel hiervoor is de geheime key die met behulp van *Diffie-Hellman* is gegenereerd.

Om de integriteit van de sessie te bewaken zal er gebruik gemaakt worden van een *session authentication code*. Ook hier is weer keuze uit verschillende algoritmes. Deze code kan door de client, maar ook de server worden gebruikt om een *Message Authentication Code* te genereren om de integriteit

en de authenticiteit van een bericht te controleren.

H.2 Authenticatie

De server en de client hebben nu een beveiligde verbinding gestart. Er is echter nog op geen enkele manier gekeken over welke gebruiker er nu probeert te verbinden. Om de identiteit van de client vast te stellen kunnen er verschillende methodieken gebruikt worden. We zullen de twee meest gebruikte hier even behandelen.

H.2.1 Public Key Authentication

Om gebruik te maken van *public key authentication* wordt er aan de server, maar ook aan de clientkant gezorgd dat er een private en een public key gegenereerd worden. De public key is, zoals de naam vermoedt, publiekelijk bekend. Deze zal verspreid moeten worden naar de andere kant van de verbinding. De private key blijft alleen bekend bij de eigenaar ervan, anders is de integriteit van de key niet meer verzekerd.

Wanneer er een authenticatiepoging wordt gedaan met public/private keys zal *OpenSSH* automatisch een *RSA*[13][7] of een *DSA*[20] authenticatie proberen. Doordat beide partijen elkaars public key kennen, maar de private verborgen hebben, kunnen ze een data versleutelen die alleen de andere partij kan ontcijferen. Er heeft dan een succesvolle authenticatie plaatsgevonden.

H.2.2 Password authenticatie

Via deze methode wordt er om het wachtwoord van de gebruiker gevraagd. Dit wachtwoord wordt dan naar de server kant verstuurd. Omdat het kanaal waarover dit gebeurt versleuteld is, zal deze niet gecompromitteerd zijn. Op de server kant zal er een gewone user authenticatie plaatsvinden. Wanneer dit een positief resultaat geeft zal er in ieder geval een geverifieerde gebruiker zijn.

H.3 Account check

Op dit moment is op een van bovenstaande manier de gebruiker geverifieerd. Nu moet nog gekeken worden of deze gebruiker wel recht heeft op shell access. Er wordt dan naar drie punten gekeken.

H.3.1 Locked

De gebruiker kan vergrendeld zijn. Dit betekent dat er wel een account is van de gebruiker, maar dat deze door een beheerder is vergrendeld. Een gebruiker mag dus om deze reden het systeem niet meer in, en zal een unlock moeten aanvragen.

H.3.2 DenyUsers

De *OpenSSH* server heeft ook een configuratie optie waar gebruikers kunnen worden opgegeven, die wel een (werkende) account hebben, maar geen toegang hebben tot *SSH*. Dit betekent dat ze zich alleen fysiek mogen aanmelden op de machine in kwestie.

H.3.3 DenyGroups

Wanneer een gebruiker zich bevindt in een groep welke in de *OpenSSH* server config staat aangegeven als een verboden groep, zal de login ook worden geblokkeerd. Men zou dus kunnen beslissen om de groep *users* toe te voegen achter de directive *DenyGroups*. Hierdoor zou het voor leden van andere groepen nog steeds mogelijk zijn om in te loggen.

H.4 Sessie voorbereiden

Nu we zeker weten dat een gebruiker ook toegang heeft op de server kunnen we voor de gebruiker een sessie gaan creëren. Dit gaat schematisch in de volgende stappen.

1. Wanneer de login plaatsvindt op een *tty* word, tenzij anders gespecificeerd in SSH config of *~/.hushlogin*, de last login en */etc/motd* geprint.
2. Als de login plaatsvindt op een *tty* word de login tijd opgeslagen.
3. Als het bestand */etc/nologin* bestaat word dit afgedrukt, en word de verbinding gesloten. Dit geldt niet wanneer de *root* gebruiker inlogt.
4. *fork()* onder user permissies.
5. Stel basis environment in. Eventueel ook dingen als X forwarding instellen.
6. Als gebruikers een eigen environment mogen instellen zal dit gedaan worden aan de hand van *~/.ssh/environment*, mits deze bestaat.

7. Verander naar de home directory van de gebruiker.

H.5 Shell

Nu alle aanvragen van de client zijn behandeld is het aan de client om een proces uit te voeren. Dit kan een speciaal commando zijn welke is opgegeven met verbinden, maar meestal zal dit de shell zijn. Beide kanten van de verbinding mogen op ieder moment data naar elkaar zenden, waardoor er een volledig interactieve omgeving is ontstaan.

H.6 Disconnect

Als de user zijn aangevraagde programma (en eventuele verbindingen met de X server) afsluit, zal de server een *exit* commando versturen naar de client, waarna de beide kanten verbinding met elkaar zullen verbreken.

Bijlage I

GNU Emacs

I.1 Introductie

Emacs[21][15] is een reeks tekst editors, waarvan in de jaren 70 de ontwikkeling is begonnen. De belangrijkste en meest gebruikte variant van *emacs* is de *GNU* versie, *GNU Emacs*. Deze tekst zal verder over deze variant gaan.

Emacs is een text editor die er naar streeft om de muis overbodig te maken. Doordat alle bediening met het toetsenbord gebeurt is het een editor die erg veel *RSI klachten* voorkomt.

I.2 Voordelen

De voordelen van Emacs zijn legio, maar een paar van de belangrijkste zullen hier beschreven worden.

Als eerste hebben we het eerder genoemde *RSI* punt. Dit word namelijk vaak veroorzaakt door het gebruik van een muis, aangezien dit (relatief) erg krampachtige bewegingen zijn.

Ten tweede is het ontzettend snel. Door het gebruik van de vele shortcuts is het voor een gebruiker van een traditionele editor (Netbeans, Eclipse, ...) onmogelijk om een Emacs of VI(M) gebruiker bij te houden.

Ook is het ontzettend lichtgewicht. Het gebruik van Emacs als editor kost bijna geen systeem resources. In de praktijk betekent het dat je systeem veel responsiever aanvoelt. Verder is de opstarttijd laag, en blijft het in gebruik erg vloeiend.

Er zijn meerdere clipboards. Ook al klinkt dit erg onzinnig is de mogelijkheid

om meerdere dingen te knippen en te plakken met een paar simpele knoppen van je toetsenbord goud waard. Het biedt erg veel mogelijkheden om grote stukken tekst of code efficiënt te herstructureren.

I.3 Shortcuts

Er is een erg grote nadruk gelegd op het gebruik van shortcuts. Hierdoor kan je ontzettend snel complexe taken uitvoeren. Deze shortcuts zijn meestal via zogenaamde *modifier keys* te gebruiken. Wanneer je in de documentatie van *Emacs* kijkt zie je deze ook veel gebruikt worden. Het gebruik van knoppen is als volgt:

Wat	Betekenis
C	Control
M	Meta (Alt)
C-f	Control + f
C-x C-f	Control + x gevolgd door Control + f
M-x sort-lines	Meta + x, gevolgd door het typen van sort-lines

Nu ben je als gebruiker in staat de documentatie en verschillende cheatsheets[5] te lezen.

I.4 Navigatie in Emacs

Om mensen toch enigzins op weg te helpen zullen hier een aantal erg veel gebruikte knoppen worden gedefinieerd.

Wat	Betekenis
C-f	Een karakter naar rechts ¹ .
M-f	Een woord naar rechts ² .
C-a	Begin van de regel.
C-e	Einde van de regel.
C-x f	Bestand openen.
C-x C-s	Bestand opslaan.
C-k	Kill 'knip' alles vanaf cursor naar einde regel.
M-Backspace	Delete een woord van cursor naar links.
M-d	Delete een woord van cursor naar rechts.
C-u	Undo.
C-y	Yank; plakken.
M-y	Cycle the kill buffer na een Yank.
C-x C-c	Emacs afsluiten

I.4.1 Configuratie

De configuratie van emacs bestaat uit het bestand `~/.emacs` in de home directory van een gebruiker. Een leuk begin van een configuratie kan het volgende zijn:

```

1 (custom-set-variables '(inhibit-startup-screen t))
2 (custom-set-faces)
3 ;; Tab completion voor buffer switching.
4 (iswitchb-mode t)
5 ;; Indent grootte
6 (setq standard-indent 4)
7 ;; Geen debiele backup files - we denken zelf na (name~)
8 (setq make-backup-files nil)
9 ;; Geen toolbar
10 (tool-bar-mode 0)
11 ;; Geen menu bar
12 (menu-bar-mode -1)
13 ;; Fullscreen wanneer er op F11 gedrukt word.
14 (defun fullscreen ()
15   (interactive)
16   (set-frame-parameter nil 'fullscreen
17     (if (frame-parameter nil 'fullscreen) nil 'fullboth)))
18   (global-set-key [f11] 'fullscreen)
19 ;; F12 voor de config file van emacs.
20 (global-set-key (kbd "<f12>")
21   (lambda() (interactive) (find-file "~/.emacs")))
22 ;; Parenthesis highlighting!
23 (show-paren-mode)
24 ;; Scrollbar rechts houden.
```

¹Dit geldt ook voor C-b, M-p en M-n, een karakter naar links, boven en onder.

²Dit geldt ook voor M-b, M-p en M-n, een woord naar links, boven en onder.

```
25 | (set-scroll-bar-mode 'right)
26 | ;; Trigger voor scrollen in een buffer.
27 | (setq scroll-margin 3)
```

Het commentaar geeft informatie over wat de regel doet. Voor een ontzettend uitgebreide website met achtergrond informatie willen we de geïnteresseerde leze doorverwijzen naar de emacs wiki[1].

Bijlage J

VIMProved

VIM is VI-Improved en is gemaakt door Bram Molenaar. De VI editor is eenvoudig te verbeteren. Het commando 'vi' start op de meeste Linux systemen de VIM editor. Door de afwezigheid van de file '~/.vimrc' start VIM in de VI compatible mode. Het enige wat dus gedaan moet worden is het aanmaken van een '~/.vimrc' bestand.

J.1 Navigatie in VIM

Om mensen toch enigzins op weg te helpen zullen hier een aantal erg veel gebruikte knoppen worden gedefinieerd.

Wat	Betekenis
i	Start insert mode; je kan typen.
ESC	Einde editing mode; je kan commando's invoeren.
l	Een karakter naar rechts ¹ .
e	Een woord naar rechts ² .
0	Begin van de regel.
A	Einde van de regel + insert mode
:e	Bestand openen.
w	Bestand opslaan.
d	Kill 'knip' alles vanaf cursor naar einde regel.
db	Delete een woord van cursor naar links.
dw	Delete een woord van cursor naar rechts.
u	Undo.
y	Yank; plakken.
q	VIM afsluiten

¹Dit geldt ook voor h, k en j, een karakter naar links, boven en onder.

²Dit geldt ook voor b, k en j, een woord naar links, boven en onder.

J.2 ~/.vimrc

Deze VIM configuratie file zorgt ervoor dat een aantal eigenschappen van VIM gaat werken.

```
1  :syntax on
2  set nocp
3  set ruler
4  set ts=4
5  set incsearch
6  set ignorecase
7  set cindent
8  set cinoptions=>0.5s,e0,n0,f0,{0,}0, 0, :s,=s,ps,ts,c3,+s,(2
   s,us,)20, 30,gs,hs
9  set cinkeys=0{,0},:,0 #,! F,o,O,e
10 set t Co=16
11 set t Sf = [[3%dm
12 set t Sb= [[4%dm
```

1. Syntax zorgt ervoor dat afhankelijk van de file extensie de juiste kleur-codering wordt gestart.
2. Ruler geeft onderaan het scherm aan in welke mode VIM momenteel staat.
3. ts=4 zet de tab-size op 4.
4. incsearch bestekend dat VIM direct gaat zoeken wanneer de eerste letter wordt weergeve. (Zoeken gaat via '/').
5. ignorecase zorgt ervoor dat tijdens het zoeken geen onderscheid tussen upper- en lower case wordt gemaakt.
6. De opties die met cin beginnen zijn voor het plaatsen van ën ën bepalen het gedrag van de 'TAB' toets.
7. De laatste drie regels zijn voor de kleuren. Het teken [[3%dm is de toetsencombinatie 'CTRL + v' en dan ESC. Als het teken dus uit 2 karakters bestaat moet deze worden vervangen door de genoemde toetscombinatie.

Bijlage K

General Public License v2

Dit gaat over versie twee van de licentie. Ook al bestaat er tegenwoordig een nieuwere (v3) is versie 2 nog steeds een belangrijke licentie. Veel grote projecten zijn onder versie 2 uitgegeven. Omdat deze licentie nog zo belangrijk is zullen we deze dan ook behandelen.

K.1 Uitwerking

Om de GPL uit te werken is er een onderverdeling gemaakt in de volgende punten

1. Welke rechten krijg je.
2. Welke plichten krijg je.
3. Wat word je verboden.
4. Overige opmerkingen.

K.1.1 Rechten

Je krijgt alle rechten om een project te kopiëren, te wijzigen en te distribueren. Dit mag met de code, maar ook met het binaire werk. Dit betekent dus dat je zonder problemen het werk mag afleiden, en een eigen tak van een project mag uitbrengen. Dit word in de community een zogenaamde *fork* genoemd. Dit geeft de nieuwe verspreider wel weer extra plichten.

Wanneer een project softwarepatenten omhelst, zal de uitgever (de maker) een onvoorwaardelijke vrijstelling voor het gebruik van de gerelateerde patenten geven. De enige manier waarop dit word ingetrokken, is wanneer een

andere partij een rechtzaak begint om de rechtmatigheid van de patenten te betwisten.

Een aanbieder is gevrijwaard van garanties. Er is dus geen mogelijkheid om de (natuurlijk) persoon aansprakelijk te stellen voor eventuele geleden schade. Aanbieders hebben echter wel het recht om contractuele verplichtingen aan te gaan en zo bepaalde diensten te verlenen. Dit stelt bedrijven in staat om diensten aan te bieden in combinatie met een open product.

K.1.2 Plichten

Wanneer een project is veranderd, en het word opnieuw gedistribueerd, dan is de nieuwe distributeur verplicht om de nieuwe, aangepaste code beschikbaar te stellen. Hier zijn verschillende eisen aan:

1. Alle aanpassingen moeten genoteerd zijn. Bovenaan een gewijzigde source file zal de naam van de persoon die de wijziging uitvoert moeten komen. Ook zal de datum van de wijziging ingevoegd moeten worden.
2. De namen van eerdere aanbieders mogen niet gebruikt worden om het product te promoten, zodat bij verminderde kwaliteit er geen image schade optreed. Er zal echter wel melding moeten blijven over het werk wat zij hebben gedaan.
3. De code moet digitaal of op cd aangeboden worden. In het laatste geval moet het tegen minimale kosten en voor minimaal 3 jaar na publicatie gelden.
4. Wanneer de code ongemodificeerd door een non-profit instelling word verspreid, zal deze de mogelijkheid hebben om source requests te verwijzen naar een bovenliggende partij.
5. Bij een nieuwe distributie van een werk zal er expliciet op de GPL licentie gewezen moeten worden. Dit betekend een vermelding in een interactief programma, een bijgevoegde licentie en references aan het begin van een source file.

Wanneer uitbreidingen aan het werk patent gerelateerde zaken bevatten, zullen deze uitbreidingen alleen legaal zijn als je dezelfde patent rechten verleend als hierboven beschreven. Dit houd dus in dat deze patenten vrij te gebruiken moeten zijn. Wanneer dit niet zo is, zal dit als een overtreding van de GPL gezien worden, waardoor alle verleende rechten ontnomen worden.

K.1.3 Verbod

Alles gerelateerd tot het beperken van de rechten tot kopiëren, modificeren of verspreiden is nadrukkelijk verboden. Herdistribueren van een afgeleid werk met extra restricties valt hier ook onder. Dit is de reden dat afgeleid werk altijd een zelfde of recentere GPL licentie moet krijgen. De recentere GPL licentie moet echter wel compatible blijven met de zelfde GPL licentie. Wordt er gebruik gemaakt van “GPL versie 2 of later” dan is dit geen probleem, maar wordt er gebruik gemaakt van “GPL versie 2” dan is dit wel een probleem. Versie 3 van de GPL is namelijk op sommige punten niet compatible, en het is dus niet zomaar mogelijk om een onder de specifieke versie 2 uitgebrachte software onder versie 3 uit te brengen.

Een ander verboden punt is het integreren van werk onder deze licentie in werk onder een andere licentie. Deze andere licentie zal andere eisen aan de distributie stellen, en deze eisen zijn expliciet verboden binnen de GPL. Integratie met ander werk wordt immers gezien als afgeleid werk, en dat moet onder eenzelfde of latere GPL licentie uitgebracht worden.

Door het bovenstaande zijn ontwikkelaars echter wel beschermd tegen misbruik van bedrijven. Ze hebben immers niet het recht om zomaar de producten te gebruiken binnen eigen werk. De enige voorwaarde is wanneer de bedrijven, of andere partijen, hun eigen werk ook weer onder voorwaarde van de GPL vrij te geven. Dit garandeert het vrije karakter van het werk.

K.1.4 Opmerkingen

Om bovenstaande verplichtingen en garanties veilig te stellen, zijn er bepaalde eisen gesteld aan de geldigheid van de licentie. De volgende punten zijn de kritieke aspecten.

Wanneer een programma aangepast of opnieuw verspreid wordt, zal dit worden gezien als het impliciet accepteren van de licentie. Bij het verspreiden zal er wel een duidelijke melding gemaakt moeten worden over het type licentie, en de rechten en plichten daarvan. Dit kan door in een GUI op een menu item te klikken, maar ook met command line switches als het een console programma betreft. Er moet door de verspreider in ieder getracht zijn de gebruiker op de hoogte te brengen.

De GPL licentie zelf mag vrij verspreid worden, maar een wijziging aan de licentie, van welke aard dan ook, zal deze licentie ongeldig verklaren.

Wanneer er door juridische of wettelijke interventie (delen van) de GPL ongeldig worden verklaard, zal de GPL zelf totaal ongeldig worden. Al het verleende copyright wordt ingetrokken binnen de regio waar deze beperkingen

gaan gelden.

Functioneel houd dit in dat de licentie zichzelf beschermd tegen juridisch en wettelijke overmacht. Wanneer een rechter bepaald dat een bepaalde clause onrechtmatig is, kan het voorkomen dat er bepaalde plichten afgeschaft worden. Dit zou betekenen dat het vrije karakter niet meer word gewaarborgd, waardoor er misbruik mogelijk gaat zijn. Een voorbeeld zou zijn dat de code ineens niet meer hoeft te worden verspreid na het aanpassen. Dit betekend dus het einde van het vrije karakter van het werk.

Dit heeft echter wel een groot nadeel. In het hypothetische geval dat de GPL wereldwijd als ongeldig word verklaard, is het gebruik van alle onder de GPL uitgebrachte werk illegaal. Theoretisch zou dit dus de complete GNU wereld kunnen vernietigen.

K.2 Licentie

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation’s software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and

disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of

a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed

it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit

geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INA-

BILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

One line to give the program's name and a brief idea of what it does. Copyright (C) ;year; ;name of author;

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use

may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Bibliografie

- [1] Various Authors. *Emacs Wiki*. <http://www.emacswiki.org/emacs-en>.
- [2] Various authors. *Man pages*.
- [3] Various Authors. *Perl Programming Documentation*. <http://perldoc.perl.org/>.
- [4] Peter Berends and Remko de Vrijer. Linux starter pack. Intern binnen Hogeschool Rotterdam verspreid, 2001.
- [5] David Cohen and Bob Rogers. *GNU Emacs Cheatsheet*. http://www.rgrjr.com/emacs/emacs_cheat.html.
- [6] Mendel COoper. Advanced bash-scripting guide. Website. <http://tldp.org/LDP/abs/html/index.html>.
- [7] Cryptool.org. Rsa cipher in 10 minutes. <http://cryptool.org/media/RSA/RSA-Flash-en/player.html>.
- [8] Peter den Brok. Computer interfacing - tircin01. Technical report, Hogeschool Rotterdam, 2011. <http://med.hro.nl/bropj/computertalen.html>.
- [9] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
- [10] Alexandre Duret-Lutz. Gnu autotools tutorial. Website, 2011. <http://www.lrde.epita.fr/~adl/autotools.html>.
- [11] Antaeus Feldspar. An explanation of the deflate algorithm. Technical report, Zlib.org, 2002. <http://zlib.net/feldspar.html>.
- [12] The Linux Foundation. Linux standard base. Website. <http://lwn.linuxfoundation.org/lsb/lsb4-resource-page#Specification>.

- [13] David Ireland. Rsa algorithm. Technical report, DI Management, 2011. http://di-mgt.com.au/rsa_alg.html.
- [14] RSA Labs. Universally unique identifier. Technical report, RSA Labs, 2007. <http://www.rsa.com/rsalabs/node.asp?id=2248>.
- [15] GNU Project. Gnu emacs. Website. <http://www.gnu.org/software/emacs/>.
- [16] Linux Kernel Project. Device list. Shipped with kernel source. Location: `file:///usr/src/linux/Documentation/devices.txt`/`usr/src/linux/Documentation/devices.txt`.
- [17] SUN. Virtualbox. Website. <http://www.virtualbox.org>.
- [18] tldp.org. The linux documentation project. Website. <http://tldp.org>.
- [19] Wikipedia. The debian almquist shell. Website. http://en.wikipedia.org/wiki/Debian_Almquist_shell.
- [20] Wikipedia. Digital signature algorithm. Website. http://en.wikipedia.org/wiki/Digital_Signature_Algorithm.
- [21] Wikipedia. Emacs wikipedia. Website. <http://en.wikipedia.org/wiki/Emacs>.
- [22] Wikipedia. Grep. Website. <http://en.wikipedia.org/wiki/Grep>.
- [23] Wikipedia. Homedirectory. Website. <http://en.wikipedia.org/wiki/Homedirectory>.
- [24] Wikipedia. Linux distro timeline. Website. <http://nl.wikipedia.org/wiki/Bestand:LinuxDistroTimeline.png>.
- [25] Wikipedia. Load. Website. http://en.wikipedia.org/wiki/Load_%28computing%29.
- [26] Wikipedia. Perl. Website. <http://en.wikipedia.org/wiki/Perl>.
- [27] Wikipedia. Regular expression. Website. http://en.wikipedia.org/wiki/Regular_expression.
- [28] Wikipedia. Symmetric key algorithm. Website. http://en.wikipedia.org/wiki/Symmetric-key_algorithm.
- [29] Wikipedia. Universally unique identifier. Website. http://en.wikipedia.org/wiki/Universally_unique_identifier.
- [30] Wikipedia. Unix shell. Website. http://en.wikipedia.org/wiki/Unix_shell#Further_reading.

Index

>, 94
>>, 94
/, 38
/dev/null, 95
/etc/HOSTNAME, 64
/etc/fstab, 38, 119
/etc/hosts, 64
/etc/resolv.conf, 64
/proc/loadavg, 62
/usr/local/src, 134
\$HOME, 133, 134
\$LOGNAME, 100
\$PATH, 133
&, 96
|, 93
~/.bash_profile, 95
~/.vimrc, 143
' , 98
1>&2, 94
2>, 94
2>&1, 95

adduser, 36
agetty, 37
array, 109
awk, 78

Bash, 37
bash, 89, 91
boot manager, 120
bzip2, 79, 86, 132

cat, 64, 79, 82, 93, 94
cd, 44
chainloader, 127
chmod, 56, 99
chown, 52

configure, 132
cp, 44
cron, 79
crontab, 79

dash, 89
date, 104
dd, 76
Debian, 15
DenyGroups, 137
DenyUsers, 137
device file, 75
devpts, 38
df, 44
diff, 80
dmesg, 115

echo, 100, 103, 106
effective uid, 53
Emacs, 139
 cheatsheet, 140
emacs, 48, 50
eSATA, 18
export, 133
EXT4, 25

false, 111
fdisk, 23, 24, 32, 120, 121
fg, 96
file, 81, 133
file mode bits, 54
find, 81, 98
flush, 76
fork, 145
free, 119
fsck, 82
fstab, 38

ftp, 68
 getent, 52
 GNU, 14
 grep, 82, 93, 94
 GRUB, 30
 grub, 121
 GUI, 14
 gunzip, 82, 83
 gzip, 82, 86, 132

 halt, 35, 36, 126
 head, 85
 historie, 24
 history, 95

 IDE, 18
 ifconfig, 64
 index, 50
 init, 35, 36, 61
 installpkg, 72, 128

 jobs, 96

 kill, 60, 61, 96
 killall, 97

 last, 52
 ldconfig, 42
 ldd, 83
 less, 41, 83, 115
 libata, 18, 19
 libusb, 19
 LILO, 30, 123
 liloconfig, 123
 ln, 83
 ls, 44, 46, 96–98
 LSB, 45

 make, 133
 makefile, 132
 man, 40, 41, 51, 78, 79, 84, 91
 mbr, 120, 127
 md5sum, 132
 mirror, 26
 mkdir, 44

 mkfs, 118
 more, 84, 93
 mount, 19, 39, 96

 nano, 48
 netconfig, 63
 nice, 84
 null, 77

 parent pid, 59
 patch, 85, 133
 perl, 45, 55, 78, 110, 111
 php, 55
 pico, 48, 49
 pid, 59
 pkgtool, 128
 POSIX, 89
 proc, 38
 ps, 58, 96
 pwd, 44
 python, 45, 55, 111

 RAM, 39
 random, 77
 rc.inet1.conf, 63
 read, 100
 real uid, 53
 reboot, 126
 removepkg, 129
 rm, 44, 95
 rmdir, 44
 root, 34
 RSI, 139

 SATA, 18
 scp, 67
 sed, 85
 seq, 106
 setup, 22
 sftp, 68
 sh, 111
 shutdown, 35
 sort, 94
 source install, 131
 ssh, 36, 38, 66, 74, 135

startx, 74
stat, 85
STDERR, 93, 94
STDIN, 93
STDOUT, 93
su, 35
swap, 39, 119

tail, 86
tar, 84, 86
telnet, 65
tmpfs, 39
top, 58, 61
touch, 107
trap, 105
tty, 37

udev, 19
uid, 53
umask, 56, 57
umount, 40
UNIX, 14
upgradepkg, 129
uptime, 61
urandom, 77
USB, 19
user management, 51
usermod, 52
uuid, 126

vi, 49, 143
vim, 48–50

w, 51, 61, 126
wget, 86
which, 133
who, 51, 126

xargs, 98
xorgsetup, 73
xwmconfig, 74

zcat, 82
zero, 77
zombie, 61