Inhoud

1 Hoofdstuk

Wat is een shell?

Tegenwoordig lijkt het alsof alle computers bestuurd worden met een muis of vingers op een touchscreen. Of dat een vooruitgang is? Op een aantal fronten absoluut, maar op andere gebieden is de ouderwetse opdrachtregel nog steeds alleenheerser. In dit hoofdstuk een stukje geschiedenis van de shell.

U leert in dit hoofdstuk:

De lagen tussen u en de hardware.

Wat een shell is.

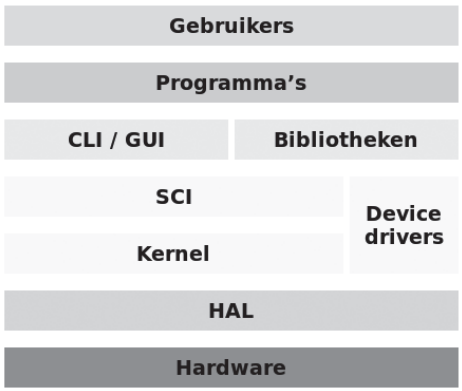
Windows-oplossing voor communicatie tussen programma's.

Waarom Bash een logische en handige keus is.

Lagen

Voor u ligt een boek over automatiseren met behulp van Bash[1]. Eigenlijk moet dit met hoofdletters geschreven worden, want oorspronkelijk is het een afkorting van **B***ourne* **A***gain* **SH***ell*. De uitleg van deze grap moet even wachten, want dat laatste woord is met de komst van Windows in de vergetelheid geraakt.

In afbeelding 1.1 staat een grove weergave van de radartjes die tezamen iets nuttigs, creatiefs of ontspannends doen. De hardware en gebruikers spreken eigenlijk voor zich. Nu zijn computers behoorlijk domme apparaten en hoewel het niet aardig is om te zeggen, geldt dat ook voor u en mij. Slechts een handjevol[2] mensen is in staat om de hardware rechtstreeks aan te sturen. De rest van ons heeft programma's nodig waarmee het genoemde nuttige, creatieve of ontspannende gedaan kan worden. Bijvoorbeeld Microsoft Excel of Libre Office Calc voor het bijhouden van een huishoudboekje, Adobe Photsohop of The Gimp voor het bewerken van foto's of een spel voor de broodnodige ontspanning.



Afbeelding 1.1 *De lagen tussen u en de fysieke hardware.*

1 Meer over de ontstaansgeschiedenis: https://en.wikipedia.org/wiki/Bash_(Unix_shell).

2 Toen computers nog zo groot waren als een huiskamer bestonden ze, maar of dat nog steeds het geval is?

De lagen in afbeelding 1.1 zijn vrij gebruikelijk in moderne besturingssystemen. Toen alles nog een stuk eenvoudiger was, kon een programma rechtstreeks op de hardware draaien. Denk bijvoorbeeld aan de eerste generatie elektronische tekstverwerkers waar een secretaresse op een (klein) scherm regels tekst kon aanpassen voordat deze op papier werden afgedrukt. Afbeelding 1.2 toont de Brother WP-1[3] die een 80C88-processor heeft (een CHMOS-versie van de 8088) om de tekstverwerker in ROM te draaien.

Afbeelding 1.2 *Brother WP-1 (Rama & Musée Bolo).*

De Brother WP-1 is echter een uitzondering. Een variant van de gebruikte processor zat in de eerste IBM-PC-computer die de basis heeft gelegd voor wat we tegenwoordig op ons bureau hebben staan. De hardware was tot veel meer in staat en daar komen de andere lagen om de hoek kijken. Die zijn nodig om een besturingssysteem zo flexibel mogelijk te maken. Voor dit boek behandel ik alleen de CLI/GUI omdat deze de buitenste laag – *shell* – van een besturingssysteem vormen. Hoewel de meeste mensen het allang vergeten zijn, waren de eerste *shells* een CLI – *Command Line Interface* of opdrachtregel. De uitvoer ging niet naar een scherm, maar naar een (matrix)printer! Starten van programma's of het onderhouden van het systeem bestond uit het intikken

3 http://www.computersammler.de/sammlung/sonstigeszubehor/brother-wp-1/.

van opdrachten, een druk op de toets Enter/Return en dan maar hopen dat alles foutloos was ingetikt…

De komst van het beeldscherm was een enorme stap voorwaarts, hoewel we dat ding tegenwoordig vanzelfsprekend vinden. Dankzij het beeldscherm konden opdrachten worden aangepast of gecorrigeerd voordat zij werden uitgevoerd. Het resultaat hoefde niet meer naar een printer en daarmee werd de eerste stap gezet naar het papierloze kantoor. Een proces dat nog steeds gaande is, zo goed is papier als uitvoer.

Al snel ontstond de muis als alternatieve invoer van het toetsenbord en kregen de beeldschermen kleur en steeds grotere resoluties. Waardoor de CLI terrein verloor aan de GUI – *Graphical User Interface* – als shell. Inderdaad, al die fraaie knopjes, menubalken, bureaublad, prullenbak en meer is niets anders dan een shell. Alles – en meer! – wat in een GUI mogelijk is, kan ook in een CLI. Sterker nog, de eerste GUI zonder (verborgen) CLI moet voor zover ik weet nog gemaakt worden.

Opdrachten

In de vorige paragraaf hebben we uitgelegd wat een shell is. De grafische varianten zoals Windows, macOS of Linux-desktopomgevingen zoals Gnome en KDE zijn prima als een muis[4] afdoende is voor het invoeren van opdrachten. Zodra het complexer wordt, schiet een grafische interface al snel tekort. In een opdrachtregel is het volgende nog vrij eenvoudig:

```
time rsync -nav --delete --password-file wdmc4.pw ../dvd rsync://wdmc4/.
```

4 Veelgebruikte opdrachten zijn meestal ook met een toetsencombinatie, bijvoorbeeld Ctrl+S voor bewaren, uit te voeren, maar onbewust wordt toch vaak de muis gebruikt.

Hier worden twee losse opdrachten (een andere naam voor programma's) gebruikt: `time` en `rsync`[5]. De eerste geeft na afloop weer hoeveel seconden de resterende opdracht heeft geduurd. In een grafisch programma is zoiets nog wel te programmeren. Voordeel van de opdrachtregel is dat de programmeurs van `rsync` over dit soort niches niet hebben hoeven na te denken.

In deze opdracht gebruiken we zes van de tig opties die de opdracht `rsync` kent. Die allemaal opnemen in een grafische interface is mogelijk, maar bijna gegarandeerd onoverzichtelijk. De optie `n` betekent dat de synchronisatie niet daadwerkelijk wordt uitgevoerd, maar dat getoond wordt wat er zou gebeuren zonder deze beveiliging. Ik maak hier voor mijn back-ups handig gebruik van om gewijzigde of te verwijderen bestanden op te sporen en deze eerst te archiveren. Daarvoor gebruiken ik een hele serie andere commando's met elk hun eigen opties. Dat allemaal in één grafisch programma verwerken en ook nog eens overzichtelijk houden, is onmogelijk. Terwijl het op de opdrachtregel nauwelijks iets voorstelt...

Windows

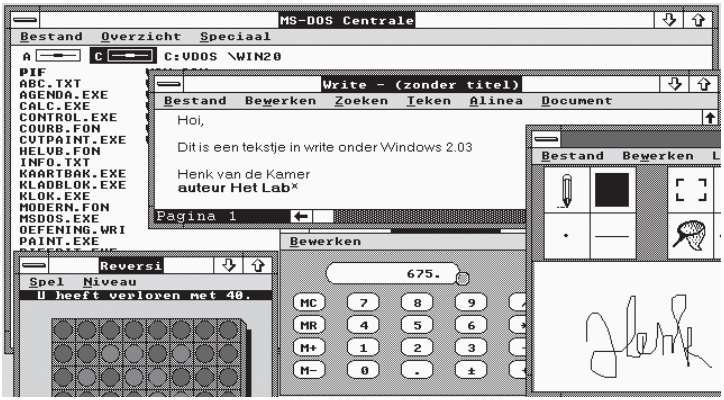
Hiervoor vergelijk ik eigenlijk appels met peren. Het voorbeeld met de opdrachtregel gebruikt twee programma's en in het back-upscript nog meer. Waarom zou het grafisch dan in één programma moeten? Dat is een goede vraag! In theorie zouden grafische programma's ook in een keten aan elkaar geknoopt moeten kunnen worden. In de praktijk blijkt dat het maken, controleren en programmeren van die verbindingen al snel te complex wordt.

Bij het lanceren van Windows 2.0 – iemand daar nog mee gewerkt? – in 1987 meende Microsoft een oplossing te hebben

5 `rsync` is een afkorting voor *remote synchronization*, synchroniseren van een directory naar een andere computer.

voor communicatie tussen twee grafische programma's: DDE[6] – *Dynamic Data Exchange*. Ik heb dit nooit gebruikt, maar volgens Microsoft zou het ook in moderne Windows-versies nog moeten werken. De meeste voorbeelden gaan uit van Microsoft Excel en plaatsen in een cel de volgende formule:

```
='Voorraad'|'bol.com'!1001004009994645
```



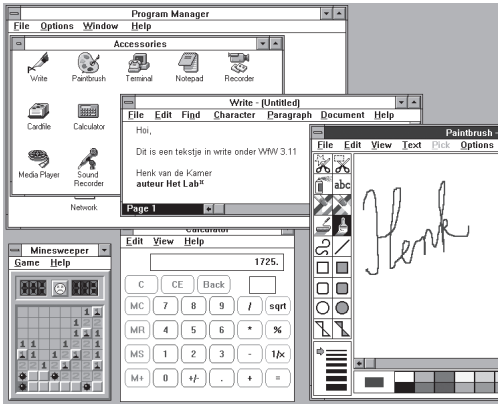
Afbeelding 1.3 *Zo zag Windows 2.0 eruit.*

Het eerste deel is de naam waarop een ander programma reageert zodra een DDE-aanroep wordt gedaan. Veronderstel dat dit programma de voorraad kan opvragen bij allerlei websites via een artikelnummer. Het tweede deel geeft de gewenste website en het daar gebruikte artikelnummer. Dit is slechts een voorbeeld, in de praktijk zal het voorraadprogramma een heel stuk slimmer moeten zijn.

Het uitwisselen van data is nuttig, maar in de praktijk willen we meer. Laten we even bij Microsoft Excel blijven. Dit programma kan op basis van data grafieken tekenen. Uiteraard kunnen we dit ook in een eigen programma doen, maar waarom het wiel

6 https://en.wikipedia.org/wiki/Dynamic_Data_Exchange.

opnieuw uitvinden? Met de komst van Windows for Workgroups 3.1 in 1993 werd COM[7] – *Component Object Model*– als oplossing voor dit probleem geïntroduceerd. Onder deze paraplu schuilt een reeks technieken en dat gaat te ver voor dit boek. Het enige wat u moet weten is dat een programma een serie functies en variabelen kan publiceren die door andere programma's gebruikt kunnen worden.



Afbeelding 1.4 *Zo zag Windows for Workgroups 3.11 eruit.*

In het verleden heb ik meerdere malen geprobeerd om COM te gebruiken, maar het is mij nooit gelukt. Het grootste probleem is geen of erg weinig informatie over de interface. En als die wel beschikbaar was, moest er vaak het nodige ingesteld worden voordat het echte werk kon beginnen. Een goed voorbeeld[8] in Delphi (een Pascal-dialect) heeft 159 regels echte code om in Excel data in te voeren, hier een grafiek van te maken, deze naar Word kopiëren om vervolgens te kunnen mailen. In hoofdstuk 8 doen we iets vergelijkbaars: uitlezen van bestandsnamen, deze in een archief plaatsen en die vervolgens mailen. Inclusief twee lege regels om het leesbaar te houden is de code 12 (!) regels groot...

7 https://en.wikipedia.org/wiki/Component_Object_Model.
8 http://edn.embarcadero.com/article/10129.

Lijm

Om bestaande programma's met elkaar te verbinden, te reageren op problemen en voor de gebruikersinterface van ons nieuwe programma is 'lijm' nodig. In het eerdere voorbeeld was dat Delphi, maar in principe kan het elke Windows-programmeertaal zijn. Van oorsprong was het bijvoorbeeld Visual Basic van Microsoft zelf.

Ook op de opdrachtregel is 'lijm' noodzakelijk en hier wordt deze meestal verwerkt in de shell zelf. In MS-DOS geeft de shell `command.com` (de `C:\>`-prompt) de mogelijkheid om opdrachten in een keten te gebruiken en om batchbestanden te verwerken. Er zijn zelfs beperkte, voorwaardelijke constructies, maar deze hangen deels af van de gebruikte versie. In MS-DOS 6.22 kan bijvoorbeeld een lijst afgedrukt worden:

```
C:\>for %i in (2 4 16) do echo %i

C:\>echo 2
2

C:\>echo 4
4

C:\>echo 16
16

C:\>
```

De uitvoer is behoorlijk verwarrend, omdat het resultaat van de `for`-lus als letterlijke opdrachten wordt doorgegeven aan een tweede instantie. Vandaar dat u steeds eerst de prompt en de opdracht ziet verschijnen, gevolgd door de uitvoer. Een `@` voor de `echo` werkt in deze MS-DOS-versie niet, omdat dan gezocht wordt naar een niet bestaande opdracht `@echo`. Toch kan de `for`-lus nuttig zijn, zoals afbeelding 1.5 laat zien.

```
C:\>for %i in (config.sys autoexec.bat) do type %i

C:\>type config.sys
device=c:\windows\himem.sys /testmem:off
device=c:\dos\emm386.exe noems
dos=high,umb
stacks=9,256
country=031,,c:\dos\country.sys
device=c:\windows\smartdrv.exe /double_buffer
devicehigh=c:\windows\ifshlp.sys
devicehigh=c:\dos\oakcdrom.sys /d:idecd001

C:\>type autoexec.bat
@echo off
path=c:\windows;%path%
set temp=c:\windows\temp
lh smartdrv.exe
lh mscdex.exe /s /d:idecd001 /m:12
lh doskey /insert

C:\>_
```

Afbeelding 1.5 *Toon de inhoud van config.sys en autoexec.bat.*

De gebruikte opdrachten `echo` en `type` in afbeelding 1.5 zijn interne opdrachten van `command.com`. In de directory `C:\DOS>` zijn alle externe opdrachten, de meeste niet erg nuttig, te vinden. Omdat niet iedereen toegang zal hebben tot een oude MS-DOS-versie om de bewering te controleren, ook even de versie in de opdrachtregel `cmd.exe` van Windows 7:

```
C:\Users\hvdkamer>for %n in (2 4 16) do @echo %n
2
4
16
```

Inderdaad, de `@` om de uitvoer echo te onderdrukken werkt nu ook op de opdrachtregel zelf. Verder zijn er parameters toegevoegd (gebruik een `for /?` voor meer informatie):

```
C:\Users\hvdkamer>for /L %n in (2 4 16) do @echo %n
2
6
10
14
```

Ziet u het verschil? De `/L` geeft aan dat de getallen als begin, stapgrootte en eind geïnterpreteerd moeten worden in plaats van drie losse getallen. Dat geeft de reeks 2, 6, 10 en 14 die getoond wordt.

GNU/Linux

Voor de komst van MS-DOS en de shell `command.com` waren er al drie Unix-shells beschikbaar[9]. De derde in dit lijstje is voor dit boek interessant: de Bourne-shell. Deze is in 1977 door Stephen Bourne in Bell Labs ontwikkeld voor Version 7 Unix.

Op 5 januari 1984 startte Richard Stallman met het GNU-project[10] met als doel het schrijven van *free software*. *Free* heeft in het Engels twee betekenissen[11]: '*for free*' of '*with little or no restriction*'. De laatste heeft *libre* als een minder bekend synoniem. In het Nederlands is de eerste gratis en de tweede vrijheid. Hoewel in het Nederlands de verwarring een stuk minder is, kennen we ook hier grensgevallen zoals vrije toegang. Tegenwoordig wordt daarmee vaak gratis bedoeld, maar van oorsprong was het toegang met inachtneming van de normale fatsoensnormen. Vrije toegang tot een natuurgebied geeft u niet het recht om afval achter te laten, dat idee. U begrijpt dat in beide talen dit juridisch niet is dichtgetimmerd en daarom gebruikt Richard Stallman de uitdrukking *'Think free as in free speech, not free beer'*.

In afbeelding 1.1 is alles tussen de hardware en de programma's onderdeel van het besturingssysteem. Het uiteindelijk doel van GNU was een vrij Unix-besturingssysteem. Om die reden begon[12] Brian Fox op 10 januari 1988 met het programmeren van Bash.

9 https://en.wikipedia.org/wiki/Comparison_of_command_shells.
10 https://en.wikipedia.org/wiki/GNU.
11 https://en.wikipedia.org/wiki/Gratis_versus_libre.
12 https://en.wikipedia.org/wiki/Bash_(Unix_shell).

Deze afkorting van *Bourne-again shell* en is een kwinkslag naar het niet vrije origineel en het concept '*born again*' (opnieuw geboren). Een ander belangrijk onderdeel – de GNU Hurd-kernel[13] – begon in 1990 aan zijn ontwikkeling.

Geheel onafhankelijk begon Linus Torvalds in april 1991 te programmeren aan wat uiteindelijk de Linux-kernel[14] (!) zou worden. Om deze te compileren gebruikte hij gcc– *GNU Compiler Collection*. Om iets nuttigs met zijn kernel te kunnen doen, gebruikte hij verder Bash versie 1.08. Omdat de GNU Hurd-kernel niet wilde vlotten – op het moment van schrijven wordt het bij mijn weten nog steeds niet op productieservers gebruikt – is het logisch dat de Linux-kernel en de GNU-tools elkaar hebben versterkt tot het besturingssysteem GNU/Linux.

Tot slot

Vrije software is tegenwoordig gemeengoed. Er zijn uitzondering zoals Microsoft Windows, maar de gebruikers weten steeds vaker de weg te vinden naar vrije software zoals Firefox, Libre Office, The GIMP, Inkscape en nog veel meer. Om deze vrijheid te bereiken is de broncode van deze programma's noodzakelijk. Hiermee kunnen via kleine aanpassingen de genoemde programma's gebruikt worden in Windows, macOS, Linux of een hele serie nauwelijks bekende besturingssystemen. En met de broncode wordt het zoeken naar fouten en achterdeurtjes mogelijk, evenals het aanpassen naar uw voorkeuren in plaats van keuze die een ontwikkelaar oplegt. Vrije software kan met recht een revolutie genoemd worden en deze is nog steeds in volle gang. Met dit boek zult u ontdekken dat Bash en de vele andere tools (grotendeels GNU) ook in het rijtje thuishoren.

13 https://en.wikipedia.org/wiki/GNU_Hurd.
14 https://en.wikipedia.org/wiki/Linux_kernel.