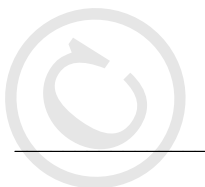


Student-notities
Linux/UNIX deel 1
Voorbeeld-hoofdstuk

07. De shell en procesbeheer

 **at computing**
The Linux/UNIXperts

Nijmegen



Copyright © AT Computing 2001-2006
Versie: 6a

Student-notities

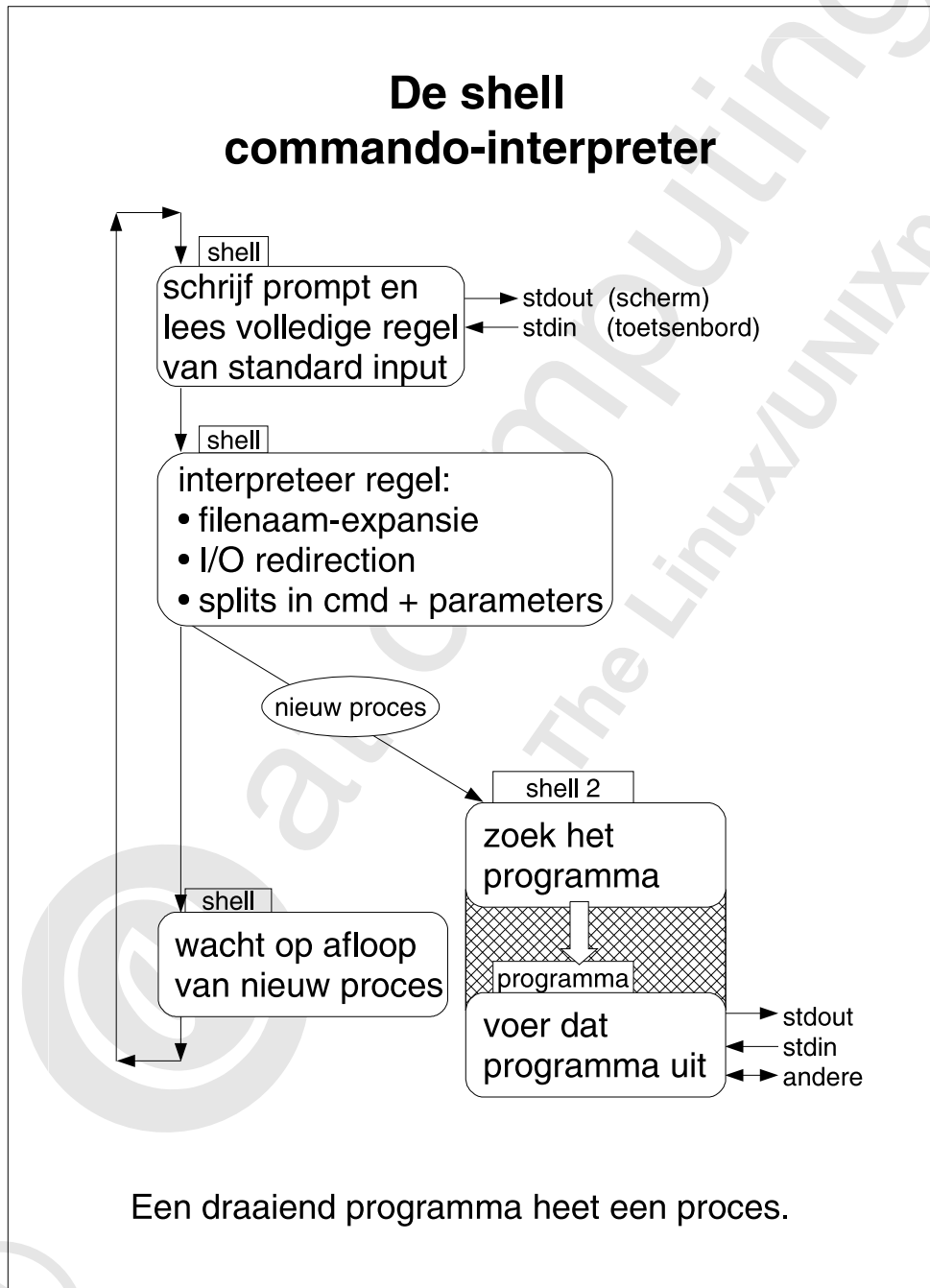
- De student-notities in dit voorbeeld-hoofdstuk zijn fragmenten uit het dictaat dat bij deze cursus wordt meegeleverd. □

Bij het inloggen van een gebruiker wordt automatisch voor de betreffende gebruiker een programma gestart dat voor de afhandeling van de commando's van deze gebruikers zorgt. Dit programma noemen we de *commando-interpreter* of de *shell*.

Overigens is het niet helemaal juist om te spreken over *de* shell, omdat de gebruiker meestal de keuze heeft uit meerdere verschillende shells. Onafhankelijk van het exacte type, doorlopen alle shell-programma's een bepaalde flow, zoals te zien is in het plaatje op de sheet.

De stappen die doorlopen worden zijn:

- De \$-prompt
De \$-prompt is in feite afkomstig van de shell: hij vraagt om een commando te geven. Systeembeheerders krijgen een aparte prompt # om hen op elk moment aan hun belangrijke verantwoordelijkheden te herinneren.
- Het gegeven commando
De getypte regel met commando wordt door de shell gelezen en opgevangen. Dit is de reden dat de shell een "commando-interpreter" wordt genoemd: hij zal uw commandoregel uitpluizen en besluiten wat er verder mee gedaan moet worden. Voordat het door u gewenste commando echter uitgevoerd wordt zal de shell eerst enkele voorbereidende dingen doen, zoals I/O redirection en filenaam-expansie ("globbing").
- Uitvoeren van het commando
Uitvoeren van een commando doet de shell meestal niet zelf. Slechts enkele commando's worden door de shell "zelf" uitgevoerd (het commando `cd` is zo'n uitzondering).
Voor alle andere commando's worden losse UNIX-programma's te hulp geroepen: door de shell wordt opdracht gegeven aan de kernel om het programma met de naam van het commando op te zoeken. Die programma's staan in het filesysteem onder directories `/bin`, `/usr/bin`, enz.
- Wachten op beëindigen van het commando
Terwijl het commando zijn werk doet, doet de shell niets. Pas als het commando eindigt, zal de shell zijn werk hervatten: hij kijkt of het commando op een "normale" manier gestopt is; is dit laatste niet het geval, bijvoorbeeld omdat het commando door het systeem geforceerd beëindigd is omdat het iets probeerde te doen wat niet kon of mocht, dan zal hij hiervan een melding geven (bijv. `Segmentation violation - core dumped`). We gaan hier verder niet op in; het betekent dat er een programmeerfout in het commando zit.
Als het programma netjes gestopt is, dan zal de shell weer terugkeren naar de eerste stap: een prompt op het scherm zetten. Vanaf dit punt herhaalt de gehele cyclus zich totdat de gebruiker als commando een `^D` typt. Dat betekent dat de shell geen commando meer te verwachten heeft en daarom stopt. Dan is men uitgelogd.



Figuur 1.

Student-notities

Het is mogelijk meerdere individuele commando's *gelijktijdig* uit te laten voeren. Zo kan men tijdrovende commando's "in de achtergrond" laten draaien, terwijl de shell-prompt beschikbaar blijft voor ander werk. De prompt blijft dan niet weg totdat het commando klaar is, maar komt meteen op het scherm te staan.

Stel er is een programma *denker* dat erg veel tijd nodig heeft om uiteindelijk een paar getallen te produceren. Men kan een dergelijk programma dan het best in de achtergrond zijn werk laten doen door te starten:

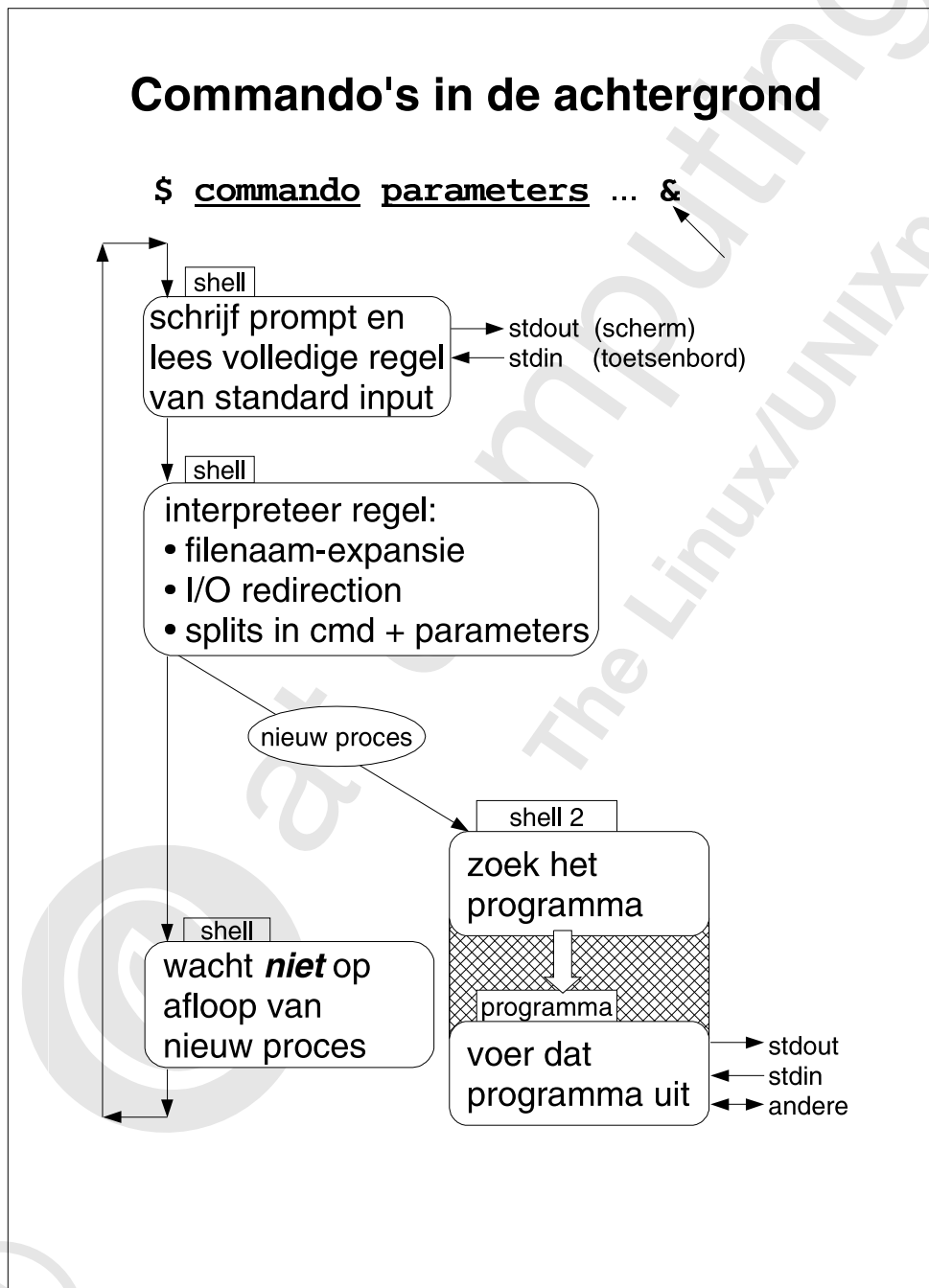
```
$ denker &
```

Het `&`-teken aan het eind van de regel betekent: "start het commando, maar wacht *niet* tot het klaar is". Nadat het nieuwe proces gestart is, accepteert de shell direct weer nieuwe commando's die dan ook meteen uitgevoerd worden.

Wanneer men een commando in de achtergrond laat draaien is het aan te raden de uitvoer van dat commando naar een file te sturen in plaats van naar de terminal. Anders kan dat de uitvoer van meerdere processen — die tegelijk in uitvoering zijn — door elkaar op de terminal verschijnen, waardoor het niet duidelijk is welke uitvoer van wie komt. Of het kan gebeuren dat er informatie naar de terminal geschreven wordt terwijl je net een nieuwe regel aan het typen was in een voorgrond proces (bijvoorbeeld een editor). Niet dat het UNIX-systeem daar problemen mee heeft, maar als mens kun je er gemakkelijk door in de war komen.

```
$ denker > denker.output &  
[1] 3257
```

Als een commando in de achtergrond gestart wordt, dan antwoordt je shell met twee getallen: het job-nummer (staat tussen `[. .]` en wordt uitgedeeld door de shell zelf), en het procesnummer ofwel *process-id*. Dit procesnummer kan gebruikt worden om het commando dat in de achtergrond draait voortijdig te volgen of te stoppen.



Figuur 2.

Student-notities

Wil je weten welke processen (voor- en/of achtergrond) er op een bepaald moment aanwezig zijn, dan kunt je het commando `ps` (process status) gebruiken.

```
$ ps
PID      TTY      TIME    COMMAND
17348    tty13   0:19    sh
23666    tty13   0:01    ps
```

Het lijstje dat `ps` default geeft, bevat voor elk proces een regel met:

- **COMMAND:** commando omschrijving
Een weergave van de commandoregel waarmee dit proces gestart werd.
- **PID:** proces-id
Het boekhoudkundige nummer dat de kernel heeft uitgereikt aan het proces (uniek binnen het systeem). Dit is hetzelfde nummer dat de shell op je scherm zet als je een commando in de achtergrond start. Als je dit getal vergeten bent, kun je het met `ps` opvragen.
- **TTY:** controlling terminal
De terminal waarmee het proces verbonden is. Dit veld correspondeert met de terminal-identificatie van het commando `who` (bijvoorbeeld: `tty13`) en de output van het `tty` commando.
In ons voorbeeld beginnen de terminal-namen met het de afkorting `tty`. Dit is niet het geval als je bent ingelogd via een netwerk-verbinding (bijv. via `telnet` of `slogin`) of als je werkt op een X-terminal; dan is de afkorting meestal `pts` (pseudo-terminal).
- **TIME:** totaal geconsumeerde CPU-tijd
Geeft aan (minuten:seconden) de tijd die de centrale processor (CPU) van de computer in totaal aan dit proces besteed heeft. Een proces dat lang aanwezig is, hoeft niet automatisch ook veel CPU-tijd te consumeren.

Het programma `ps` wil wat betreft vlaggetjes en layout van de output nogal eens variëren tussen de verschillende UNIX-versies. Sommige versies van `ps` proberen zelfs een selectie te maken tussen “interessante” en “niet interessante” processen. In bovenstaande tabel worden dan `sh` en `ps` als “niet interessant” beschouwd omdat men wel weet dat er een shell aanwezig is. En dat `ps` zichzelf ook op de lijst zet, is wel leuk maar niet erg informatief.



Proces-manipulatie - 1

\$ ps (process status) geeft een lijst van de processen van huidige terminal of window:

PID	TTY	TIME	CMD
21602	pts/12	00:00:04	bash
21739	pts/12	00:00:00	ps

ps heeft veel vlaggetjes:

- afhankelijk van de concrete UNIX/Linux
- globaal gezien een paar stijlen:
 - POSIX (de meeste UNIXen)
 - BSD UNIX (NetBSD, FreeBSD, OpenBSD, Mac OS X)

De **ps** op Linux is POSIX-stijl maar met allerlei BSD-stijl aanvullingen.

Ander systeem? De manualpagina raadplegen!

POSIX-stijl vlaggetjes onder andere:

-f	(full) Meer gebruikers-info
-l	(long) Meer systeembeheerders-info
-e	Alle processen op het systeem
-u user	Alle processen van die gebruiker
-t term	Alle processen aan dat terminalwindow
-p pid	Info over dat proces

Veel gebruikte combinaties:

```
$ ps -ef
```

```
$ ps -fu carla
```

Figuur 3.

Student-notities

Het proces dat in de voorgrond loopt, kan meestal gestopt worden door vanaf de terminal `^C` te geven. Dit geldt niet altijd; sommige processen negeren dit, omdat de programmeur dat negeren speciaal heeft ingebouwd. Een voorbeeld is de `vi`-editor, die niet stopt maar alleen het lopende subcommando afbreekt.

Ook processen die in de achtergrond lopen reageren niet. Als je ze wilt stoppen dan moet dat op een andere wijze. Het benodigde UNIX-commando daarvoor heet `kill` en wordt inderdaad gebruikt voor datgene wat zijn naam suggereert: het "doodslaan" van processen die om de een of andere reden losgeslagen zijn en geforceerd gestopt moeten worden.

De syntax van het commando `kill` is als volgt:

```
kill -s SIGnaam PID ...
kill -signalnummer PID ... (ouderwets)
```

De processen met de proces-id nummers die als argumenten worden opgegeven, krijgen van `kill` een signal, een soort seintje. Je gebruikt natuurlijk `ps` om achter de betrokken procesnummers komen.

De signals hebben in het UNIX-jargon een nummer en een naam. De meest gebruikte signals zijn:

Signal:	Jargon-naam:		Vanaf terminal veroorzaakt met:
1	hangup	SIGHUP	verbreken van de verbinding
2	interrupt	SIGINT	typ <code>^C</code>
3	quit	SIGQUIT	typ <code>^\</code>
9	kill	SIGKILL	(niet mogelijk)
15	terminate	SIGTERM	(niet mogelijk)

Het `interrupt`-signal kan ook aan een voorgrondproces gestuurd worden zonder het commando `kill` te gebruiken. Het wordt gegenereerd door de daarvoor gedefinieerde toetsaanslag `^C` en daaraan is ook zijn naam ontleend. Dat dit signal daarnaast ook via het `kill`-commando kan worden gegenereerd is bijzaak.

Met het `terminate`-signal correspondeert geen toetsaanslag; het is primair bedoeld om van buitenaf aan het proces door te kunnen geven dat verdere activiteit ongewenst is en dat je graag wilt dat het proces stopt. Dit `SIGTERM`-signal wordt ook gegeven als men het `kill`-commando gebruikt *zonder* signalnummer als parameter.

Het sturen van signals naar processen van andere gebruikers is zinloos: deze signals worden door het systeem niet doorgegeven.

Proces-manipulatie - 2

Een seintje ("signal") naar een proces sturen:

```
$ kill -s signal pid [pid...]
```

Ouderwets: `kill -signal pid [pid...]`

Je kunt alleen signals sturen naar je eigen processen.

Mogelijke signals onder andere:

- 1 **HUP** (hangup) "Je window is weg", wordt automatisch verstuurd aan alle processen die aan een terminalwindow hangen als dat window gesloten wordt. Bij serverprocessen, met de hand verstuurd: "Lees je configuratie-file opnieuw".
- 2 **INT** (interrupt) Hetzelfde als `^c` intypen voor het voorgrond-proces.
- 9 **KILL** Dodelijk, proces stopt abrupt.
- 15 **TERM** (terminate) Vriendelijk verzoek om te stoppen, applicatie kan eerst nog zaken afronden. Default signal als je geen signal opgeeft: `kill 24731`

Signalnamen: hoofd- en kleine letters hetzelfde.

Een lijst van alle mogelijke signals opvragen:

```
$ kill -l (letter ell)
```

Figuur 4.

Student-notities

Met het woord *job* worden alle processen bedoeld die tegelijkertijd met één commandoregel worden gestart (bij meerdere processen gebeurt dat via pipes). Elke *job* die in voor- of achtergrond loopt, wordt door de shell geregistreerd. Deze registratie bevat de commandoregel, het process-id, en de status (running of stopped).

Je shell deelt aan iedere job (commandoregel) een eigen volgnummer uit: het job-number. Dat job-number is bedoeld voor boekhouding binnen de shell; het staat los van het process-id, want dat dient voor de systeem-wijde boekhouding van de kernel. Bovendien zijn deze nummers niet één-op-één, want een job kan uit meerdere processen bestaan.

Als een job in de achtergrond wordt gestart toont de shell het job-number (tussen [] haken), gevolgd door het proces-id nummer. Jobs die in de achtergrond lopen, mogen geen invoer lezen van het toetsenbord. Als de processen binnen zo'n job daartoe toch een poging wagen, worden ze automatisch in de toestand "stopped" gezet.

Jobs en processen

Eén commandoregel wordt één *job*:

```
$ ls (voorgrond)
$ ping & (achtergrond)
$ who | sort > uit (voorgrond)
```

↙ ↘
twee processen, één job

Twee nummeringen: proces- en jobnummer:

```
$ who | sort >uit &
[1] 25493
$
```

↙ ↘
job-id process id van
 achterste proces
 in de job (sort)

Alleen de voorgrond-job kan bij het toetsenbord:

```
$ mutt &
[1] 26201
$
[1] Stopped
$
```

↙ ↘
probeert van standard input
(toetsenbord) te lezen, dus

Het job-nummer is alleen bekend binnen dezelfde shell, dus bijv. niet in een ander window.

Figuur 5.

Student-notities

Met het `jobs`-commando (een shell-builtin) zie je welke jobs er op een gegeven moment actief zijn:

```
$ du /usr > lijst &
[1] 1998
$ du -a /usr | sort | pr | lpr &
[2] 2044
$ jobs
[1] + Running      du /usr > lijst
[2] - Running      du -a /usr | sort | pr | lpr
```

Een job die in de achtergrond loopt kan tijdelijk “bevroren” worden. Dit bereik je door het commando `stop` te geven met het jobnummer — voorafgegaan door een %-teken — als parameter:

```
$ stop %2
$ jobs
[1] + Running      du /usr > lijst
[2] - Stopped      du -a /usr | sort | pr | lpr
```

Merk op dat een gestopte job geen processortijd meer kan gebruiken. Zo'n job is tijdelijk opzij geschoven.... Een gestopte jobs kan later weer in de achtergrond doorgestart worden met het commando `bg` gevolgd door het jobnummer:

```
$ bg %2
```

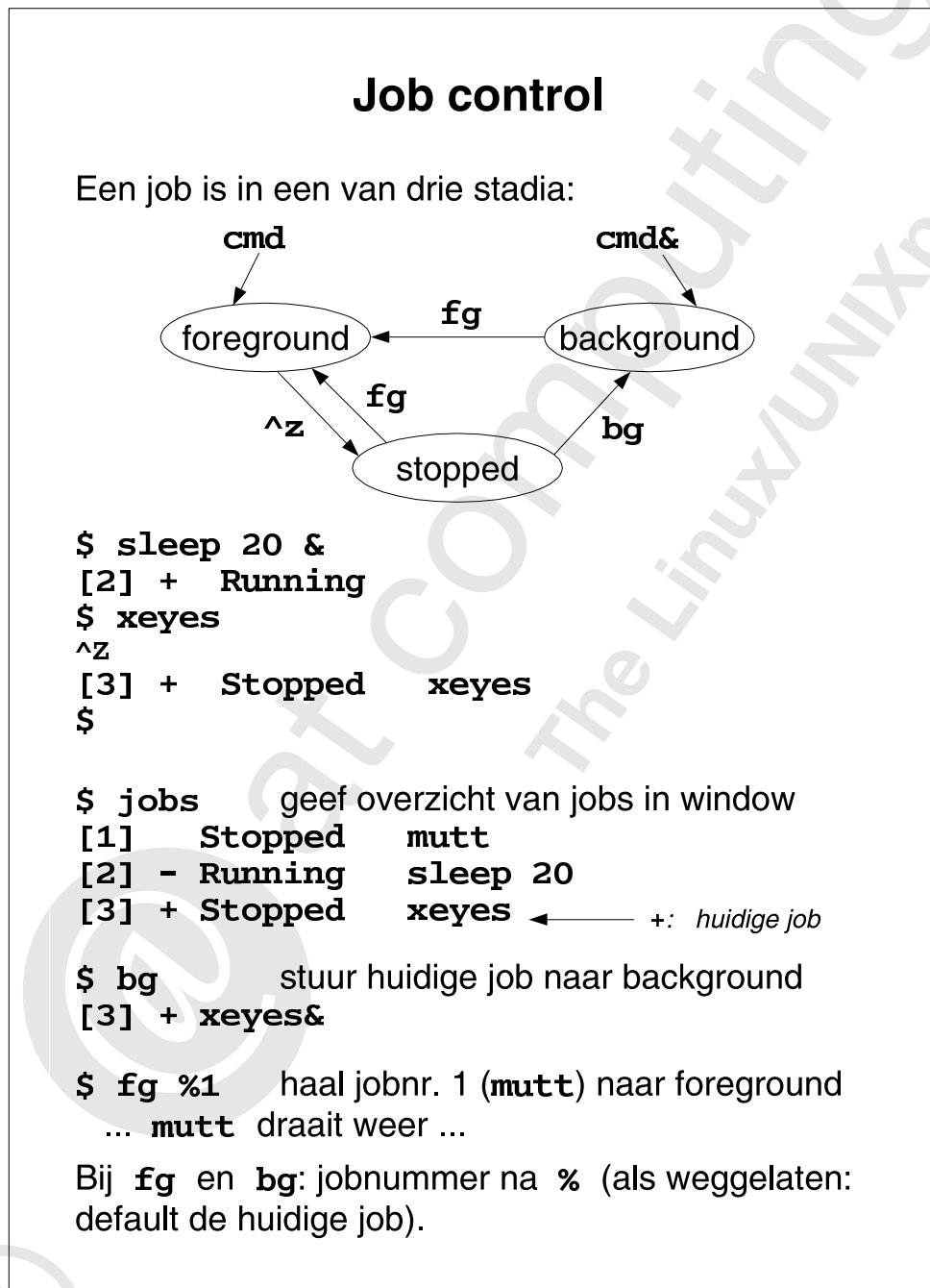
Ook een voorgrond job kan tijdelijk “bevroren” worden. Dit doe je met de toetsaanslag `^Z`:

```
$ vi grotefile
.....
.....
^Z
Stopped
$ jobs
[1]  Running      du /usr > lijst
[2] - Running      du -a /usr | sort | pr | lpr
[3] + Stopped      vi grotefile
```

Het doorstarten van een gestopte job in de voorgrond doe je met het commando `fg` (foreground).

```
$ fg %3
vi grotefile
..... (de edit-sessie wordt in voorgrond vervolgd)
```

Wordt bij commando's die een jobnummer nodig hebben dit laatste weggelaten, dan werken deze commando's op de *huidige job* te herkennen aan het plusteken + in de uitvoer van het `jobs`-commando.



Figuur 6.

Student-notities

In het resterende deel van dit hoofdstuk komen *shell-variabelen* ter sprake, evenals de grondbeginselen van *shell-scripts*. Verder behandelen we kort de commando's `nohup` en `nice`, en gaan we in op de verschillen tussen de soorten shells en de gebruikers-profile.



at computing
The Linux/UNIX experts