Korte uitleg: Welk bestand wordt uitgevoerd bij een commando?

Nadat de shell alle handelingen uit *Korte uitleg: Wat doet de shell met mijn commandoregel?* heeft gedaan blijft er een "eenvoudig commando" (simple command) over. Dat heeft de vorm:

```
VAR1=VAL1 .... commando argument1 argument2 ....
```

Voor de betekenis van de eventueel aan commando voorafgaande environment variable toekenningen zie *Korte uitleg: Environment variabelen (zoals PATH, DISPLAY, LANG, ...)*. Meestal zijn die er niet, maar ook als ze er zijn gaat het hier om het stuk:

```
commando argument1 .....
```

Waarschijnlijk verwacht je dat commando hier een programma of zo is, dat uitgevoerd gaat worden. Dat klopt en we gaan nu bekijken welk programma uitgevoerd gaat worden. Tenslotte zijn de meeste programma's bestanden en welk bestand wordt nu geladen en als proces gestart?

Er zijn twee mogelijkheden: commando bevat één of meer tekens `/` of niet.

Als er geen `/` in het eerste woord van het eenvoudige commando staat gaan we naar **stap 1** hieronder, anders naar **stap 4**.

Stap 1

Als er geen `/` in commando staat gaat `bash` eerst kijken of het een ingebouwd commando (shell builtin) is. Zo ja, dan wordt dat uitgevoerd. Voorbeeld:

```
cd test
```

Het is dus niet handig om een programma de naam van een shell builtin te geven! Als het inderdaad om een shell builtin gaat zijn we hier ook aan het einde van de zoekaktie. Zo niet dan gaan we verder met **stap 2**.

Stap 2

Als het geen shell builtin is wordt gekeken of commando als Alias is gedefinieerd. Wat is dat nu weer? De shell heeft een lijst van Aliases. Je kunt die lijst beheren met de builtin commando's `alias` en `unalias`. De vorm van een Alias is:

```
Aliasnaam='Aliastekst'
```

Ons commando (het eerste woord van het eenvoudige commando) wordt opgezocht in die lijst van Aliasnamen. Als het gevonden wordt, wordt het commando vervangen door de Aliastekst. Een voorbeeld:

```
alias l='ls -alF'
```

Als het commando nu is:

```
l test
```

wordt het:

```
ls -AlF test
```

Dit is duidelijk bedoeld voor mensen die haast hebben. Een commando dat lang is en dat je vaak gebruikt kun je zo tot een enkele letter terugbrengen. Ook voor mensen die hun MS-DOS gewoontes niet kunnen afleren is er redding:

```
alias dir='ls -l'
```

Om je huidige Aliaslijst te zien:

```
alias
```

Wil je al bij het opstarten van `bash` Aliassen toevoegen of verwijderen, maak dan het bestand `~/.alias` en zet daarin de `alias`/`unalias` commando's. Omdat in `~/bashrc` staat:

```
test -s ~/.alias && . ~/.alias || true
```

wordt dat dan aan iedere `bash` opstart toegevoegd.
Na het uitvoeren van deze Alias vervanging hebben we weer een eenvoudig commando (met in het eerste woord een `/` of niet). We gaan daarmee terug naar het begin. Hieruit blijkt dat het resultaat van een Alias vervanging weer een Alias zou kunnen zijn. Die wordt dan ook vervangen, behalve als het eerste woord van een Alias vervanging hetzelfde is als de oorspronkelijke Alias (dat zou en eeuwige lus worden), dan stopt de Alias vervanging. **Stap 2** eindigt dus altijd met een (eventueel gewijzigd) eenvoudig commando. Is er nog steeds geen / in het eerste woord dan gaan we naar **stap 3**, anders naar **stap 4**.

Stap 3

We hebben nu een woord dat een uit te voeren programma definiëert met de naam van te laden bestand. Maar waar is dat bestand? Daar komt de omgevingsvariabele (environment variable) `PATH` in het spel. (Voor environment variable zie *Korte uitleg: Environment variabelen (zoals PATH, DISPLAY, LANG, ...)*). Hier heb je een voorbeeld:

```
henk@boven:~> echo $PATH
/home/henk/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/kde3/bin:/home/henk/.local/bin
henk@boven:~>
```

Het is een lijst van directories gescheiden door `:` . De shell gaat nu van voor naar achter door die lijst en kijkt in iedere directory of een bestand van de gezochte naam bestaat. De volgorde van de directories in de lijst is dus belangrijk omdat bestanden met dezelfde naam in verschillende directories kunnen voorkomen. Het kan dus gebeuren dat het verkeerde bestand wordt uitgevoerd! Je kunt uiteraard zelf de inhoud van `PATH` wijzigen. Wees daar voorzichtig mee. Vooral als je `root` bent. Die heeft een andere `PATH`:

```
boven:~ # echo $PATH
/sbin:/usr/sbin:/usr/local/sbin:/root/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/kde3/bin
boven:~ #
```

Als er niets wordt gevonden geeft de shell een foutmelding:

```
henk@boven:~> aap
If 'aap' is not a typo you can use command-not-found to lookup the package that contains it, like this:
    cnf aap
henk@boven:~>
```

Feitelijk is de melding: "kan het programma niet vinden", maar er wordt meteen een paar suggesties gedaan. Je kunt testen wat er gevonden zal worden zonder het programma uit te voeren:

```
henk@boven:~> which ls
/usr/bin/ls
henk@boven:~>
```

Dit is tevens het einde van deze tak van het zoekproces. Het bestand is gevonden (of niet).

Stap 4

Als er een / in commando staat wordt het commando als het pad naar het uit te voeren bestand gezien. Dat kan natuurlijk een absoluut pad of een relatief pad zijn (zie *Korte uitleg: De enkelvoudige hiërarchische bestandsorganisatie (directory tree, absolute and relative pathes)*). Voorbeeld van een absoluut pad:

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

en van een relatief pad:

```
./script
```

Als dat niet bestaat:

```
henk@boven:~> /usr/bin/aap
bash: /usr/bin/aap: Bestand of map bestaat niet
henk@boven:~>
```

Ook hier zijn we aan het eind van het zoekproces. Het bestand is gevonden (of niet).

Als het bestand gevonden is wordt aan de Kernel gevraagd om er een proces van te maken. Dan is er natuurlijk nòg een kontrole: mag de gebruiker die het proces wil starten het bestand wel uitvoeren? Met andere woorden er moet een passend x-bit aanstaan (zie *Korte uitleg: Wie mag wat met welk bestand*).

Er is een hulpmiddel om te vertellen wat het resultaat van bovenstaand zoekproces is voor een commando:

```
henk@boven:~> type firefox
firefox is /usr/bin/firefox
henk@boven:~> type l
l is een alias voor 'ls -alF'
henk@boven:~> type type
type is een ingebouwde shell-functie
henk@boven:~>
```