Korte uitleg: Wat doet de shell met mijn commandoregel?

Het onderstaande is heel erg Bash gericht, maar geldt i.h.a. ook voor andere shells. Vooral als het om "begrip" gaat.

Iedere regel die aan de shell wordt gegeven (door intikken in een CLI sessie, of door het uitvoeren van een script) wordt eerst op allerlei manieren door de shell geïnterpreteerd en dan pas komt het commando tot uitvoering. Veel "Hij doet het niet" (lees "Hij doet niet wat ik dacht dat hij zou doen") situaties komen voort uit het feit dat wij mensen gedachtensprongen maken terwijl een computer programma als de shell domweg stap voor stap al zijn acties uitvoert. Wij mensen moeten dus proberen om niet over stappen heen te springen, maar ze zelf ook allemaal te maken om te begrijpen wat er gebeurt.

Dit is zo belangrijk dat ik er nog even op doorga. Als ik intik:

```
ls -l *.html
```

dan ziet het `ls` programma dus nooit die `*.html`. Want de shell gaat dat eerst expanderen. Daarna wordt het `ls` programma aangeroepen met bijv.:

```
ls -l aap.html noot.html
```

Het `ls` programma weet dus niet wat er oorspronkelijk is ingetikt. Het krijgt gewoon vier argumenten voor zijn kiezen: `ls`, `-l`, `aap.html` en `noot.html`.

De shell aan de andere kant weet niets van `ls`. En dus ook niet of `ls` zo'n rij bestandsnamen wel begrijpt. De shell expandeert en zegt tegen de Kernel: start `/usr/bin/ls` met de volgende argumenten: `ls`, `-l`. `aap.html` en `noot.html`.

Als je dit echt begrijpt wordt het begrijpend lezen (en schrijven) van een commando al een stuk makkelijker.

De shell splitst een regel op in woorden. Daarvoor zien we de volgende definities:

witte ruimte (blank)

: Een spatie (space) of tab (tab).

woord (word)

: Een opeenvolgende rij tekens die door de shell als een eenheid wordt gezien. Ook "token" genoemd.

Allerlei tekens hebben voor de shell een bijzondere betekenis. Die moet je dus allemaal kennen, want anders gebruik je ze terwijl je denkt dat ze gewoon

"zichzelf" betekenen. Dit is de lijst:

metacharacter

> Een teken dat woorden scheidt. Eén van de volgende: `| & ; ( ) < >` space tab

control operator

> Een token (let op, dus een woord) dat een controle functie uitvoert. Eén van de volgende: `|| & && ; ;; ( ) | |&` newline

Let op!, als je de speciale betekenis van zoiets dus wilt onderdrukken moet je "quoten" of "escapen". Zie daarvoor *Korte uitleg: Haakjes en Aanhalingstekens*.

Constateer ook dat het einde van een regel (in een script of met de Return toets) een betekenis heeft.

Er zijn ook "reserved words". Dat zijn dus woorden/tokens die je niet zelf moet gebruiken. Dit geldt alleen op bepaalde plaatsen: als het het eerste woord is van een eenvoudig commando, en op de derde plek in een case of for commando. Het zijn:

> `! case do done elif else esac fi for function if in select then until while { } time [[ ]]`

Maar dat is lang niet alles. De shell bekijkt de aldus verkregen tokens en als dat "gewone" woorden zijn, kijkt de shell weer naar speciale tekens daarbinnen om allerlei acties uit te voeren. Daardoor wordt een token als `*.html` geëxpandeerd tot twee nieuwe woorden: `aap.html` en `noot.html.` Hieronder vallen:

- De pathname expansion (pattern matching) tekens: `*`, `?` en `[` (met bijbehorende `]`);
- De `$` als inleiding tot diverse constructies zoals Parameter expansion, Command substitution en Arithmetic expansion.

Die worden allemaal hieronder behandeld.

Ook moeten allerlei quote en escape tekens op een gegeven moment weer verdwijnen want als ik doe:

```
ls -l 'bestandsnaam met spaties erin'
```

dan moet `ls` aangeroepen worden met drie argumenten: `ls`, `-l` en `bestandsnaam met spaties er in`. Dus zonder die `'` en `'`, want de bestandsnaam zelf bevat die `'` en `'` niet.

In de `man` pagina van `bash` staan al de behandelingen waar een regel doorheen gaat op een rijtje. Het zijn er zeven (7!). Een echte hersenkraker. Ik geef ze

hier in het Engels, tenslotte vind je nadere informatie ook eigenlijk altijd in het Engels:

> The order of expansions is: brace expansion, tilde expansion, parameter, variable and arithmetic expansion and command substitution (done in a left-to-right fashion), word splitting, and pathname expansion.

I.h.a. verandert het aantal woorden/tokens niet door een expansie, behalve bij brace expansion, word splitting (logisch hè) an pathname expansion (die kennen we al, het woord `*.html` levert twee woorden op in ons voorbeeld).

Zoals al aangegeven, het is niet eenvoudig om die zeven stappen allemaal te voorzien als je iets intikt, maar we zullen ze toch allemaal even langsgaan. Sommigen zul je in de praktijk weinig last van hebben (en ze nog minder gebruiken).

Brace expansion

Daar wil ik eigenlijk weinig over zeggen. Het is een typische `bash` feature. Het werkt zuiver op de tekst en weet al helemaal niets van commandos. Kennelijk iemands hobby geweest. Als je er ooit last van denkt te hebben, zet het dan uit met

```
set +B
```

Wel is belangrijk dat het met braces, dus `{` en `}` werkt. En die worden ook voor andere dingen gebruikt. Kun je dus `{ }` niet verklaren in iets wat je ziet, hopelijk denk je dan aan deze mogelijkheid.

Tilde expansion

Als een woord begin met `~` dan wordt de rest van de characters van dat woord tot aan de eerste `/` (en die `/` moet er zijn) gezien als een loginnaam/username. Dus bij `~abc/Documenten/brief` wordt aangenomen dat `abc` een gebruikersnaam is. Het geheel wordt dan vervangen door het pad van de home directory van die gebruiker. Dus in de meest normale situatie wordt `~abc/Documenten/brief` vervangen door `/home/abc/Documenten/brief`.

Een speciaal (maar het meest gebruikte) geval is als de `~` alleen staat: `~/Documenten/brief`. Dan wordt `~` vervangen door de home directory van de huidige gebruiker, jijzelf dus. Vooral in scripts is het handig om `~/` te gebruiken om een pad naar iets binnen in je eigen home directory aan te geven.

Parameter expansion

De volgende stap is het vervangen van allerlei variabelen aangegeven met bijv `$1`, `$AAP`, `${NOOT %%./*}` door hun waarde op dat moment.

```
henk@boven:~> TESTDIR="test"
henk@boven:~> cd ~/${TESTDIR}
henk@boven:~/test> pwd
/home/henk/test
henk@boven:~/test>
```

Dus `cd ~/${TESTDIR}` wordt eerst `cd /home/henk/${TESTDIR}` en dan `cd /home/henk/test` en dat wordt uitgevoerd.

Command substitution

De constructie `$(.....),` die we kennen uit *Korte uitleg: Haakjes en Aanhalingstekens*, wordt vervangen door de utvoer van het omsloten commando.

Arithmetic expansion

De contructie `$(( .... ))`, die we ook kennen uit *Korte uitleg: Haakjes en Aanhalingstekens*, wordt vervangen door het resultaat ven de berekening.

Word splitting

Nu worden de woorden van elkaar gescheiden. Let op, als bij de voorgaande bewerkingen een resultaat van nul tekens oplevert, levert dat geen woord op. Het levert wel een woord op als zo'n leeg woord tussen quotes staat. Dus:

```
henk@boven:~/test/bestanden> WOORD=""
henk@boven:~/test/bestanden> ls -l verf.gif $WOORD verf.jpeg
-rw-r--r-- 1 henk wij 157943 10 mei 20:39 verf.gif
-rw-r--r-- 1 henk wij 157943 10 mei 20:38 verf.jpeg
henk@boven:~/test/bestanden> ls -l verf.gif "$WOORD" verf.jpeg
ls: kan geen toegang krijgen tot : Bestand of map bestaat niet
-rw-r--r-- 1 henk wij 157943 10 mei 20:39 verf.gif
-rw-r--r-- 1 henk wij 157943 10 mei 20:38 verf.jpeg
henk@boven:~/test/bestanden>
```

In het eerste geval reduceert `$WOORD` tot niets. In het tweede geval reduceert `"$WOORD"` tot een lege string. Maar een bestand van die naam bestaat niet. Daar krijgen we dus een foutmelding over.

Pathname expansion

Wordt ook wel File name expansion of Globbing genoemd.

Ieder woord wordt bekeken op het aanwezig zijn van de tekens `*`, `?` en `[`. Zo ja, dan wordt aangenomen dat het woord een "pattern" (patroon) is. Dat patroon wordt vervangen door een alphabetisch gesorteerde lijst van bestandsnamen die aan dat pattern voldoen (die bestanden moeten dus bestaan)

We hebben de volgende bestanden:

```
henk@boven:~/test/bestanden> ls
ASH.html  index.html  verf.gif  verf.jpeg  WARNING_README.txt
henk@boven:~/test/bestanden>
```

Een `*` betekent: iedere string, inclusief een lege string, dus

```
henk@boven:~/test/bestanden> ls *
ASH.html  index.html  verf.gif  verf.jpeg  WARNING_README.txt
henk@boven:~/test/bestanden> henk@boven:~/test/bestanden> ls verf*
verf.gif  verf.jpeg
henk@boven:~/test/bestanden> henk@boven:~/test/bestanden> ls *v*
verf.gif  verf.jpeg
henk@boven:~/test/bestanden>
```

Een `?` betekent één character:

```
henk@boven:~/test/bestanden> ls
ASH.html  index.html  varf.gif  verf.gif  verf.jpeg  WARNING_README.txt
henk@boven:~/test/bestanden> ls v?rf.gif
varf.gif  verf.gif
henk@boven:~/test/bestanden>
```

En `[...]` betekent één van de tekens uit het rijtje:

```
henk@boven:~/test/bestanden> ls
ASH.html  index.html  varf.gif  verf.gif  verf.jpeg  vurf.gif  WARNING_README.txt
henk@boven:~/test/bestanden> ls v[ae]rf.gif
varf.gif  verf.gif
henk@boven:~/test/bestanden>
```

Er zijn meer mogelijkheden tussen die `[...]`, bijvoorbeeld `[0-9]` betekent alle tekens van 0 t/m 9:

```
henk@boven:~/test/bestanden> ls
1aap  2noot  ASH.html  index.html  verf.gif  verf.jpeg  WARNING_README.txt
henk@boven:~/test/bestanden> ls [0-9]*
1aap  2noot
henk@boven:~/test/bestanden>
```

Als laatste worden alle `\`, `'` en `"` die zelf niet gequote worden en die niet het resultaat zijn van één van de bovengenoemde expansions, verwijderd.

Als je de `man` pagina van `bash` bekijkt zul je zien dat er allerlei uitbreidingen op mijn beschrijving zijn. Ook is het mogelijk om allerlei handelingen net even anders te doen door opties van `bash` te zetten. Het bovenstaande is het tipje van de sluier, maar aan de andere kant bevat het de meest gebruikelijke zaken. Als je zelf scripts gaat schrijven probeer dan een consequente wijze van gebruik van de bijzondere tekens te maken. Laat dit een onderdeel van je programmeerstijl zijn. Zo heb je de beste kans dat je later je eigen scripts nog weer kan lezen en begrijpen.