Korte uitleg: Interactief systeem gebruik met een Shell

Het is leuk om een draaiend systeem te hebben, maar we willen dat ook gebruiken. Hoewel bij het opstarten van het systeem al allerlei processen worden gestart Hebben we als gebruiker (en als beheerder) behoefte aan het geven van opdrachten aan het systeem als wij dat willen. In de tijd dat Unix achtige systemen werden "uitgevonden" waren er alleen "terminals" op teken/character basis. Dus een soort tikmachines met een toetsenbord en oorspronkelijk met papier om de tekens op te laten zien.

Om dat te laten werken is er dus (alweer) een programma nodig dat leest vanaf de terminal, iets doet met wat het heeft gelezen, dan iets vraagt aan de Kernel en dan de resultaten op het papier zet. Omdat het voor de gebruiker een soort van omhulling van de Kernel is, kregen deze programma's de soortnaam Shell (schelp). De Kernel zit veilig binnen en wij hameren er van buiten op.

De eerste shell was de Bourne shell `sh`. Een uitgebreidere versie is de Korn shell `ksh`, maar hij is in kleine details verschillend. Een veel meer afwijkende shell is C shell `csh`, waarvan de syntax is geïnspireerd door de programmeertaal C. In een poging tot standaardisering onstond de POSIX shell. Tegenwoordig is dat `sh`. Dus `sh` is niet meer de Bourne shell maar de POSIX shell. Bij het ontstaan van GNU Linux was ook een shell nodig en zoals alles voor GNU nieuw is gemaakt om claims te voorkomen, maakte men ook een nieuwe Bourne again shell (een woordgrapje op de naam van Richard Bourne, maar ook een eerbewijs) `bash`.

Let op! De meeste mensen gebruiken op Linux `bash`, maar soms hebben ze de foute gewoonte om hun `bash` scripts door `sh` te laten uitvoeren. Dat gaat meestal goed, maar niet altijd.

Wat doet de shell dus voor je in de meest eenvoudige vorm? Stel je tikt

```
ls -l
```

dan vraagt de shell aan de Kernel om een kindproces te starten met als uit te voeren programma `/usr/bin/ls` en als argumenten `ls` en `-l`. (Als het niet helemaal duidelijk is waarom van `ls` gemaakt wordt `/usr/bin/ls`, lees dan *Korte uitleg: Welk bestand wordt uitgevoerd bij een commando?* met uitleg van de `PATH` process environment variable.) Als de Kernel aan dat verzoek voldoet gaat het programma `ls` draaien en de uitvoerteksten komen zichtbaar op de terminal. So simpel is het.

Maar de shell kan meer. Er zijn allerlei zaken aan toegevoegd, die programmeren mogelijk maken. Alle basis constructies van programmeren zijn er: if-then-else, loop, parameter gebruik, enz. Je kunt dus zeer ingewikkelde commando's/statements intikken, maar als het erg ingewikkeld wordt en als je

zoiets ingewikkelds vaker wilt doen, is het handiger om de zaak in een bestandje te zetten en dat bestandje ter interpretatie aan de shell aan te bieden. Eén commando met hele grote gevolgen.

Zulke programma's worden niet gecompileerd zoals bij C, Fortran of Cobol, maar worden direct vanaf de broncode (de source) geïnterpreteerd. Nadeel is dat dat niet zo efficiënt is, maar voordeel is dat het makkelijk is te wijzigen en direct weer te proberen. Ze worden scripts genoemd en omdat dit scripts voor een shell zijn: shell scripts, beter: bash scripts.

Kortom een shell is niet anders dan een programma dat aan de gebruiker een Command Line Interface biedt om het systeem te gebruiken.