Linux CLI: Werken met commando's

D. Leeuw

13 mei 2025

1 Over dit Document

1.1 Leerdoelen

Na het bestuderen van dit document heeft de lezer kennis van:

- hoe commando's gebruikt worden
- hoe je kan zien of een commando succesvol verlopen is
- het gebruik van ; | && ||
- de commando's: type, which, xargs

Dit document sluit aan op de volgende onderdelen van de LPI:

- LPI Linux Essentials 010-160 - 6.2.1 Command Line Basics (weight: 3)

1.2 Voorkennis

Voor een goed begrip van dit document wordt er van de lezer verwacht dat die weet:

- wat de shell is
- wat een variabele is
- hoe er omgegaan moet worden met man-pages

2 Commando's

Commando's of opdrachten aan de shell hebben een vaste vorm (syntax). De syntax ziet er zo uit:

```
commando<spatie>optie(s)<spatie>argument(en)
```

De spaties zorgen ervoor dat de shell weet wanneer een volgend deel begint, voor de eerste spatie staat het commando, daarna volgen er geen of enkele opties en tot slot zijn er geen of enkele argumenten. Bijna alle commando's houden deze syntax aan, hoewel er ook uitzonderingen zijn.

Opties zorgen ervoor dat een commando zich anders gaat gedragen, dan zonder opties. Zoals bij het `ls` commando, waar zonder opties het alle bestanden in een directory laat zien, maar met de `-l` optie zorgt voor meer informatie over de bestanden in de directory.

Argumenten vertellen een commando waarop de actie uitgevoerd moet worden. Een argument kan een bijvoorbeeld een bestand zijn:

```
$ cd ~
$ ls -l .bash_history
```

Bij het `cd` commando is de tilde (~) het argument en bij het `ls -l .bash_history` commando is `-l` de optie en het bestand `.bash_history` is het argument. Dit laatste commando toont de informatie over het `.bash_history` bestand.

2.1 Waar zijn de commando's?

Een gemiddeld Linux systeem bevat heel veel commando's en dat is omdat er in de Unix-wereld twee filosofieën zijn die bij elkaar aansluiten de eerste is het KISS-principe. KISS staat voor Keep It Simple, Stupid en is oorspronkelijk afkomstig uit de US Navy. De tweede is Small is Beautiful en die is afkomstig uit de Economie.[1]

Je komt op UNIX-achtige systemen dan ook vele kleine commando's tegen die één ding goed doen. Dit heeft een aantal voordelen. Omdat ze maar één ding doen is de code simpel en is dus code waarin de programma's geschreven zijn makkelijker te controleren op fouten. Een ander voordeel is dat niet iedereen in elk programma weer dezelfde code hoeft te herhalen maar gebruik kan maken van iemand anders zijn programma. De totale hoeveelheid code is daardoor klein, een compleet Linux systeem met grafische interface kan geïnstalleerd worden op een 15GB disk, zonder grafische interface past het zelfs op een 5GB disk. De laatste reden is dat wij als gebruikers vaak zonder te programmeren al heel complexe dingen met Linux kunnen doen omdat we al die kleine commando's aan elkaar kunnen knopen waarmee complexe dingen te doen zijn.

Het nadeel van heel veel kleine programma's is dat je het overzicht snel kwijt kan raken, zeker ook omdat vele commando's soms cryptische afkortingen hebben. Zo staat `ls` voor list. De oorspronkelijke ontwikkeling van Unix werd gedaan op systemen met toetsenborden die niet zo ergonomisch zijn als tegenwoordig. Ze hadden toetsen die je met enige kracht moest indrukken en na een dag programmeren hadden de programmeurs Ken Thompson en Dennis Ritchie en hun team vaak zere knokkels door overbelasting. Door commando's zo kort mogelijke namen te geven verminderden ze het aantal toetsaanslagen. Vandaar de vaak korte commando namen.

Type

```
$ ls
```

[1] Small Is Beautiful: A Study of Economics As If People Mattered door E. F. Schumacher

en je zal een aantal blauwe directories op je scherm zien verschijnen. Als je nu

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

typt dan verschijnen er allemaal groene commando's op je scherm, of beter ze scrollen van je scherm af in kolommen, omdat het er heel veel zijn. Het past niet op je scherm. Als we de hele lijst willen zien dan zullen we gebruik moeten maken van een programma de uitvoer van `ls` opdeelt in pagina's die zoveel regels bevatten dat ze het scherm vullen. Een programma dat dat doet heet `more`. De kunst is nu om de uitvoer van `ls` te koppelen aan `more` en daarvoor is er de pipe, `|`, of de pijp. Het pipe-character koppelt twee commando's aan elkaar:

```
$ ls /usr/bin/ | more
```

met de spatie-balk kan je nu pagina voor pagina bekijken en met de letter q verlaat je `more`. Nu is `more` wel heel simpel en kan het alleen dat wat je nu gezien hebt. Makkelijker zou het zijn als je omhoog en omlaag door de commando's kan gaan, en misschien zelfs wel zou kunnen zoeken in zo'n lange lijst. Dat kan ook, daarvoor hebben we de opvolger van `more` die meer kan en `less` heet want less is more.

```
$ ls /usr/bin/ | less
```

Nu kan je met de spatie-balk door de pagina's gaan, met de pijltjes omhoog en omlaag per regel door de lijst gaan, met PgUp en PgDn per pagina omhoog en omlaag gaan en met / kan je zoeken. Type maar eens als je in `less` zit

```
/less
```

Zo kom je bij het eerste commando uit dat less in de naam heeft. Met n kan je zoeken naar het volgende (next) commando dat ook less bevat, en zo vderer . Ook hier weer is de q-toets de manier om `less` te verlaten.

We hebben nu gezien dat heel veel commando's terug te vinden zijn in de `/usr/bin/` directory. Maar dit is maar één plek waar commando's te vinden zijn. Commando's vind je terug in de `bin/` en `sbin/` directories. We gebruiken hier bewust een meervoud omdat we deze directories op verschillende plekken kunnen vinden. Je bent in de `/usr` directory al de `bin/` directory tegen gekomen. De `bin/` directories bevatten commando's die door iedereen gebruikt kunnen worden. De `sbin/` directories zijn voor de commando's die alleen toegankelijk zijn voor de systeembeheerder.

Naast in `/usr/` vind je ook `bin/` en `sbin/` directories in de `/` en de `/usr/local/` directory. Al deze locaties hebben een andere functie:

- / is de root van het systeem en de `bin/` en `sbin/` directories bevatten daar de commando's die nodig zijn voor het opstarten van het systeem en zijn afkomstig van de distributie.
- `/usr/` bevat (s)bin directories die de commando's bevatten voor normaal gebruik van het systeem en deze commando's zijn afkomstig van de distributie.
- `/usr/local/` bevat (s)bin directories die commando's bevatten die door de systeembeheerder geïnstalleerd zijn (gecompileerd)
- `/opt/` bevat subdirectories met daarin (s)bin directories per geïnstalleerd softwarepakket dat niet door de distributie is meegeleverd, maar later vanaf een pakket is geïnstalleerd.

Om te bepalen welke directories gebruikt worden voor het zoeken naar een commando is er een variabele aanwezig in de shell en die variabele heeft de logische naam PATH. Type maar eens:

```
$ echo $PATH
```

Dit levert een output op die er ongeveer zo uit ziet: `/usr/local/bin:/usr/bin:/bin` (dit kan per systeem verschillen). Voor een gebruiker met dit PATH wordt er voor een commando eerst gezocht in `/usr/local/bin/`, daarna in `/usr/bin/` en als laatste in `/bin/`.

Als we willen weten waar een commando vandaan komt, dan kunnen we `which` gebruiken:

```
$ which ls
```

Met het commando su kan je als een andere gebruiker inloggen (switch user). Type eens:

```
$ su - root
```

geef het root wachtwoord en type

```
# echo $PATH
# exit
```

Na het su commando moet je het password van de root gebruiker geven dat je ingesteld hebt tijdens de installatie. Je zult nu zien dat het PATH van de root gebruiker ook de `sbin/` directories bevat. De root gebruiker heeft dus veel meer commando's tot zijn beschikking dan een normale gebruiker.

Als we de shell ook op andere plekken willen laten zoeken naar commando's dan moeten we de shell-variabele aanpassen. Het aanpassen van de variabele doen we met `export`:

```
$ echo $PATH
$ PATH=".:${PATH}"
$ export PATH
$ echo $PATH
```

met deze opdracht hebben we de . directory toegevoegd aan de PATH variabele. Als we een commando aanroepen en het komt voor in de directory waar we op dat moment in staan dan zal het dat commando uitvoeren.

2.2 Shell builtin commando's

Het `cd` commando wordt gebruikt om van directory te wisselen. Met `which` kunnen we zien waar een commando zich bevindt. Gebruiken we nu

```
$ which cd
```

Dan komen we tot de ontdekking dat we geen resultaat terug krijgen. Het commando bestaat en toch is het niet te vinden in ons PATH. Met `cd` en nog een paar commando's is wat speciaals aan de hand. De shell heeft een aantal ingebouwde (builtin) commando's. We kunnen met `type` zien of het een builtin commando is:

```
$ type cd
```

Via `man` kan je de `bash-builtins` pagina opvragen om te zien welke ingebouwde commando's bash heeft.

2.3 Error codes

Stel dat we een commando hebben uitgevoerd en we krijgen niets terug, hoe weten we dan zeker dat het goed gegaan is? Dat weten we omdat een commando ook een exit-code terug geeft. Een exit-code van 0 betekend dat alles goed gegaan is en alles boven 0 dat er iets fout gegaan is. De man-page van het programma kan je vaak meer vertellen over welke exit-code wat betekent als je er niet uit komt met de beschrijving van de error. Type het volgende

```
$ touch hello.txt
$ cat hello.txt
$ echo $?
$ cat Hello.txt
$ echo $?
```

na de eerste `echo $?` krijg je een 0 en na de tweede `echo $?` een 1. Dat komt omdat het eerste commando uitgevoerd kan worden en het tweede niet. Het bestand Hello.txt (met een hoofdletter) bestaat niet.

Het $-teken betekent dat we te maken hebben met een variabele. Het vraagteken is een speciale variabele die de shell gereserveerd heeft om de exit-code in op te slaan.

3 Commando samenstellingen

Met het indrukken van de Enter-toets geef je aan dat je klaar bent met het invoeren van het commando en dat de shell de gegeven opdracht moet gaan uitvoeren. Officieel hoort er na een commando een ; te komen om aan te geven dat het commando klaar is. Na een ; kan er dan nog een commando komen. Om met deze oefening aan de slag te gaan moeten we eerst een extra packet installeren:

```
$ sudo apt-get install ncal
```

Nu het `ncal` packet geïnstalleerd is, heb je de beschikking over het `cal` commando.

```
$ cal 10 2019;
$ cal 10 2019; cal 10 2020
```

Op de eerste regel van dit voorbeeld hebben we 1 commando dat afgesloten wordt door een ;. Op de tweede regel hebben we 2 commando's achter elkaar gescheiden door een ;.

Het kan ook zijn dat we het tweede commando pas willen uitvoeren als het eerste commando goed verlopen is, dan willen we dus een beslissing nemen op basis van de error-code van het eerste commando. De shell kent hiervoor de && constructie:

```
$ cal 10 2019 && cal 10 2020
$ cal 13 2019 && cal 10 2020
```

Het omgekeerde is ook mogelijk, namelijk dat we een commando uitvoeren als een voorgaand commando niet goed is verlopen. De shell kent hiervoor de || constructie:

```
cal 10 2019 || cal 10 2020
cal 13 2019 || cal 10 2020
```

3.1 pipe

Op het Linux systeem hebben we vele kleine commando's die één ding goed doen, door de data van het ene commando door te geven aan het andere commando kunnen we complexe dingen doen. Data doorgeven van het ene

commando aan het andere doen we met het pipe-character. Op je toetsenbord is dat het | teken. Met `apt-cache` kunnen we door de package database van Debian zoeken naar tools die we zouden willen gebruiken. Een packet dat je misschien zou willen gebruiken heet `ncal`. We gaan eerst de informatie van `ncal` opvragen:

```
$ apt-cache show ncal
```

Door de output van `apt-cache` via `pipe` door te sturen naar een ander commando kunnen we de output veranderen. We kunnen de output bijvoorbeeld doorsturen naar `tr`, een commando dat translaties (veranderingen) doet:

```
$ apt-cache show ncal | tr '[a-z]' '[A-Z]'
```

Bij deze laatste variant zijn alle kleine letters zijn vervangen door hoofdletters. De output van het `apt-cache` commando is aangenomen door het `tr` (translate) commando en die heeft de reeks a-z vervangen door A-Z. Dus elke a is een A geworden, elke b een B, etc.

3.2 xargs

We hebben met | (pipe) gezien dat we de output van het ene commando kunnen voeren aan een volgend commando. Het kan ook voorkomen dat je de output van een commando wilt gebruiken als een argument van een volgend commando. Dat kan niet met pipe, met de pipe geef je data door, daarvoor heb je `xargs` nodig. Het commando `xargs` neemt de output van een commando en maakt er argumenten van:

```
echo "Aap Noot Mies" | mkdir
echo "Aap Noot Mies" | xargs mkdir
```

4 Opdrachten

1. Wat doet een optie bij een commando?
2. Wat doet een argument bij een commando?
3. Hoe weet je of een commando foutloos is verlopen?
4. In mijn home-directory wil ik een directory maken `Cursus` met daarin de directories `Shell` en `Commandos`. Pas als de directory `Cursus` is aangemaakt zonder fouten wil ik dat de overige twee directories worden aangemaakt. Welk commando moet ik geven?

5. Het commando `file` laat zien wat voor bestandstype een bestand is. Het gebruikt een bestandsnaam als argument. Hoe zorg je ervoor dat je van alle bestanden in je home-directory te zien krijgt wat voor bestandstype het is?

Index