Linux: Werken met bestanden op de CLI

D. Leeuw

20 mei 2025
1.0.0

1 Over dit Document

1.1 Leerdoelen

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

- hoe er gewerkt kan worden met bestand en directories
- de commando's: touch, cat, mkdir, rmdir, rm
- het werken met redirects (onleidingen): >, », <, «
- de betekenis van stdin, stdout en stderr

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

- LPI Linux Essentials 010-160 - 6.2.3 Using Directories and Listing Files (weight: 2)
- LPI Linux Essentials 010-160 - 6.2.4 Creating, Moving and Deleting Files (weight: 2)

1.2 Voorkennis

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

- kennis heeft van de werking van de shell
- kennis heeft van de werking van commando's, opties en parameters
- kan werken met man-pages

2 Werken met bestanden

Als alles een bestand is in Linux dan is het werken met bestanden het belangrijkste wat er is. Dit hoofdstuk gaat je dan ook de basisbeginselen bijbrengen van het werken met bestanden.

2.1 Directories

Om data op een computer te structureren is het handig om bestanden te verdelen over directories. Directories zijn ook bekend als mappen en folders. Wij zullen alleen nog spreken van directories omdat dat binnen de Unix-wereld de meest gebruikte term is.

Een directory maak je aan met het commando `mkdir`:

```
$ mkdir LinuxCursus
```

Met `ls` kan je controleren of de directory ook daadwerkelijk aangemaakt is.

Je kan ook meerdere directories tegelijk aanmaken door ze als een lijst op te geven, gescheiden door spaties:

```
$ mkdir Aap Noot Mies
```

Soms wil je ook een heel pad gelijk aanmaken met:

```
$ mkdir Boom/Roos/Vis/Vuur
```

gaat dat niet lukken, want de Boom directory bestaat niet. Gelukkig kan je aan `mkdir` en optie meegeven die vertelt dat `mkdir` ook alle onderliggende directories moet aanmaken:

```
$ mkdir -p Boom/Roos/Vis/Vuur
```

Gebruik `ls` om te controleren dat alle directories aanwezig zijn.

Tot slot wil je ook instaat zijn om directories weg te gooien. Met `rmdir` kan dit als de directory leeg is.

```
$ rmdir Aap Noot Mies
```

gooit keurig alle aangemaakte directies weg. Controleer dit met `ls`. Maar doen we:

```
$ rmdir Boom
```

dan krijgen we een error melding, want de `Boom` directory is niet leeg. We zullen dus eerst alle andere subdirectories moeten weggooien, te beginnen met `Vuur`, dan `Vis`, dan `Roos` en tot slot kunnen we pas `Boom` weggooien. Gebruik `rmdir` om alle directories **behalve** Boom te verwijderen. De `Boom` directory moet dus blijven bestaan. Het `cd` commando gebruiken we om van directory te wisselen (cd staat voor **c**hange **d**irectory). We kunnen aan `cd` een relatief of een absoluut pad meegeven. Een absoluut pad is het complete pad vanaf de root-directory:

```
$ cd /home/dennis
```

Waarbij je "dennis" vervangt door je eigen gebruikersnaam zorgt ervoor dat je in je eigen home-directory komt te staan. Als je een relatief pad gebruikt betekent dit dat je niet het hele pad meegeeft, maar een stukje. Bijvoorbeeld:

```
$ cd LinuxCursus
```

Je geeft niet het complete pad `/home/dennis/LinuxCursus` op maar slechts het deel `LinuxCursus`, wat dus relatief is ten opzichte van de home-directory waarin je al staat.

3 Bestanden maken en stdin, stdout en stderr

Zorg dat je in de directory LinuxCursus staat en type:

```
$ touch hello.txt
```

Na de Enter lijkt er helemaal niets te gebeuren. Dit is met de meeste Linux commando's het geval. Als het goed gegaan is dan laten ze niets weten, een beetje als "geen nieuws, is goed nieuws". Doen we een `ls` dan zien we dat er een bestand is aangemaakt dat hello.txt heet.

Met `touch` kunnen we dus bestanden aanmaken, dit zijn lege bestanden. Type maar eens:

```
$ cat hello.txt
```

dan zal je zien dat er weer niets op je scherm verschijnt. En dat is goed! Het `cat` commando plaatst de inhoud van een bestand op het scherm en daar we een leeg bestand hebben opgevraagd is wat er op het scherm komt dus niets en omdat dat succesvol is verlopen hoeft `cat` ook geen foutmelding te laten zien en met de wetenschap dat geen nieuws, goed nieuws is is `cat` klaar.

In een vorig hoofdstuk hebben we met `echo` tekst naar het scherm geschreven en nu hebben we met `cat` een bestand op het scherm afgebeeld. Vanuit Linux gezien is dat niet helemaal correct geformuleerd. Zowel `echo` als `cat` schrijven naar de "standaard output" en in de terminal is het scherm de standaard output. De standaard output wordt vaak afgekort als stdout.

We kunnen de standaard output ook omleiden (redirect) naar bijvoorbeeld een bestand:

```
$ echo 'Ik werk met Linux' > hello.txt
```

We zien nu dat de zin die we met echo afbeelden niet meer op het scherm verschijnt. Hij is verdwenen en er lijkt weer helemaal niets gebeurd te zijn. Als we nu

```
$ cat hello.txt
```

doen dan zien we waar onze zin is gebleven. Hij is in hello.txt terecht gekomen. We hebben de stdout van `echo` in hello.txt gestopt.

Laten we dat nog eens doen:

```
$ echo 'Hello World!' > hello.txt
```

Doen we een `cat` van hello.txt dan zien we dat onze eerste zin verdwenen is en er alleen nog 'Hello World!' in hello.txt zit. We hebben kennelijk ons bestand overschreven met een nieuwe inhoud. We kunnen ook tekst toevoegen aan een bestand:

```
$ echo 'Ik werk met Linux' >> hello.txt
```

door gebruik te maken van het dubbele groter dan teken voegen we een regel toe aan het eind van het bestand. De oude regel zie je met `cat` als eerste en daaronder komt onze nieuwe regel.

Zou er als we een stdout hebben ook een standaard input (stdin) zijn en kunnen we daar dan van lezen? Ja, die is er! Als je typt:

```
$ cat < hello.txt
```

dan vertellen we eigenlijk dat `cat` de standaard invoer (stdin) op het scherm moet afbeelden. Dit is meer typen dan alleen `cat hello.txt` dus dit gaan we zo nooit gebruiken. Hoe we standaard input wel kunnen gebruiken is door bijvoorbeeld aan de shell te vertellen dat hij vanaf de standaard input moet lezen tot hij een markering tegen komt en daarna moet stoppen.

```
$ cat <<EOF
> Hallo beste mensen
> dit is een stukje tekst
> dat uit meerdere regels bestaat
> EOF
```

Je ziet dat dan standaard invoer afgebeeld wordt op het scherm zodra deze de EOF tegen komt. We hebben met `«EOF` tegen cat gezegd dat hij van standaard input moet blijven lezen tot hij de letters EOF (End Of File) tegen komt. Daarna doet `cat` nog steeds waar het goed in is, namelijk het afbeelden op de standaard output. We kunnen natuurlijk ook de standaard output van `cat` omleiden naar een bestand:

```
$ cat <<EOF >hello.txt
> Hallo beste studenten
> dit is een stukje tekst
> dat uit meerdere regels bestaat
> EOF
```

We vertellen `cat` dus dat hij moet lezen van standaard input totdat hij EOF tegen komt en dat zijn standaard output geredirect moet worden naar het

bestand hello.txt. Nadat wij de EOF hebben ingetypt verschijnt de tekst niet op het scherm, maar zit in hello.txt wat we met `cat` kunnen controleren.

Naast de standaard input en standaard output is er ook nog standaard error (stderr), waar de foutmeldingen naartoe gaan. Laten we eens een fout maken door een niet bestaan bestand aan `cat` te geven:

```
$ cat Hello.txt
```

Je krijgt nu een foutmelding dat Hello.txt niet bestaat. Linux is case-sensitive dus hello.txt is niet hetzelfde als Hello.txt. De standaard output, input en error zijn genummerd in Linux. Stdin is 0, stdout is 1 en stderr is 2. Nu we dit weten zouden we het volgende kunnen doen:

```
$ cat Hello.txt 2> Hello_error.txt
```

We zien nu geen foutmelding meer op ons scherm en hebben de stderr omgeleid (redirect) naar het bestand Hello_error.txt. Doen we nu een

```
$ cat Hello_error.txt
```

dan zien we dat de foutmelding daar is opgeslagen.

Deze vormen van het redirecten (omleiden) van data stromen gebeurt in Linux heel vaak. Programma's schrijven bijvoorbeeld de fouten die ze tegen komen naar een log bestand. Dit doen ze door de stderr te redirecten en de error regels toe te voegen aan het bestand, bijvoorbeeld 2»error.log. Als ze dit doen met een datum- en tijdmelding dan kan je heel makkelijk problemen opzoeken.

Wij hebben nu geleerd om bestanden aan te maken en om invoer en uitvoer te redirecten.

4 Kopieëren, verplaatsen, verwijderen

Om bestanden te kopieëren gebruiken `cp` van het Engelse copy:

```
$ cp hello.txt Boom/hello.txt
```

We kunnen ook gelijk de naam veranderen als we dat willen:

```
$ cp hello.txt Boom/Hallo.txt
```

Om bestanden verplaatsen gebruiken `mv` van het Engelse move. Het verschil met copy is dat een bestand niet meer op de oorspronkelijke plek terug te vinden is. Bij move heb je dus maar 1 bestand na de handeling, na copy heb je 2 bestanden.

```
$ mv Boom/Hallo.txt .
```

Het `mv` commando kunnen we ook gebruiken om bestanden van naam te veranderen:

```
$ mv Hallo.txt hallo.txt
```

Voor het weggooien van bestanden gebruiken we `rm` van remove.

```
$ rm hallo.txt
```

Omdat alles een bestand is op een Linux systeem zijn ook directories bestanden, speciale bestanden, maar toch bestanden. We hebben al gezien dat we met `rmdir` lege directories weg kunnen gooien. Zouden we nu `rm` kunnen gebruiken om ook directories weg te gooien. Ja, dat kan, maar ook hier geldt dat de directory leeg moet zijn.

```
$ rm Boom
```

geeft weer een foutmelding. Het systeem zegt tegen ons dat `Boom` een directory is. Als we tegen `rm` vertellen dat hij de zaken recursive weg met gooien, dan zal de hele boomstructuur wegggegooid worden:

```
$ rm -r Boom
```

De `-r` optie verteld aan rm dat hij vanuit de diepste plek in de Boom directory moet beginnen met bestanden (en directories) weg te gooien net zo lang tot hij de directory Boom weg kan gooien. In ons voorbeeld is het niet veel dat `rm` te doen heeft, maar je zal niet de eerste zijn die met `-r` meer weggooid dan de bedoeling was en dat tot de ontdekking komt dat een Linux systeem geen undelete kent. Terug halen van data is er dan ook niet bij.

5 Vragen en opdrachten

- Ik wil een nieuw bestand hebben met de naam "mijn eerste bestand.txt". Dit bestand moet in een nieuw te maken directory komen. De directory heeft de naam "test_map". In zowel het bestand als de directory namen komen de quotes niet voor. Zorg ervoor dat het bestand pas aangemaakt wordt als het gelukt is om de "test_map" aan te maken. Alle commando(s) staan op 1 regel. Hoe ziet de opdrachtregel eruit?
- Gebruik `echo` om de regel "Ik heb een eerste bestand gemaakt" **toe te voegen** aan het hierboven gemaakte bestand. Hoe ziet de opdrachtregel eruit?
- Hernoem het hiervoor gemaakte bestand naar "MijnEersteBestand.txt" en zorg ervoor dat deze in dezelfde map blijft. Hoe ziet de opdrachtregel eruit?

- Verwijder de map “test_map”. Hoe ziet de opdrachtregel eruit?

Index