Linux CLI: Zoeken en vinden

D. Leeuw

13 mei 2025
1.0.0

1 Over dit Document

1.1 Leerdoelen

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

- het terug vinden van bestanden op disk
- het zoeken en vervangen van tekst in bestanden
- globbing
- regular expressions
- de commando's: grep, find, sed

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

- LPI Linux Essentials 010-160 - 6.3.2 Searching and Extracting Data from Files (weight: 3)

1.2 Voorkennis

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

- werken met een editor op Linux (`vi`/`vim`)
- omgaan met quoting en escaping en wat dat betekent
- het documentatie systeem (`man`)

2 Zoeken en vinden

In dit hoofdstuk ga je leren hoe je op een Linux systeem naar bestanden kan zoeken. Ook ga je leren hoe je in bestanden kan zoeken naar patronen, waarbij woorden patronen kunnen zijn, maar er kan veel meer. Dat alles komt in dit hoofdstuk aan bod.

Om de opdrachten uit dit hoofdstuk te kunnen maken is het van belang om de volgende commando's uit te voeren:

```
$ cd $HOME
$ mkdir Zoeken_en_vinden
$ cd Zoeken_en_vinden
$ touch plaatje.jpg
```

```
$ touch Plaatje.png
$ touch Slaatje.txt
$ mkdir -p Muziek/Rammstein/
$ touch Muziek/Rammstein/mutter.mp3
$ mkdir -p dit/is/een/heel/diepe/directory
$ mkdir -p dit/is/ook/een/heel/diepe/directory
```

```
cd ~
mkdir ZIB
echo "Een commando op Unix \
systemen is vaak klein \
en simpel en doet 1 \
ding goed." > ZIB/kleineCommandos.txt
echo "Door verschillende van deze tools \
te combineren kunnen complexe taken \
volbracht worden." > ZIB/complexeTaken.txt
echo "De documenten voor dit boek \
houden hetzelfde principe aan." > ZIB/boek.txt
echo "Het zijn korte stukken tekst \
die gezamenlijk een compleet boek vormen. \
We noemen dit het \
small-is-beautiful-principe." > ZIB/small_is_beautiful.txt
echo "We kunnen dit op boeken, \
 tekst en software code toepassen." > ZIB/toepassen.txt
```

3 Zoeken naar bestanden

3.1 Globbing

Globbing is het gebruiken van wildcards. Wildcards zijn bijvoorbeeld de asterisk (*) en het vraagteken (?). Een asterisk staat voor geen of meer characters en een vraagteken staat voor één character. Zo kunnen we ontbrekende characters invullen of aanvullen als we niet meer helemaal zeker weten hoe een bestand heet. Stel dat we weten dat we een document gemaakt hebben met de naam `Plaatje` maar dat we niet meer weten of dat een jpg, png of gif plaatje is. Dan kunnen we een overzicht van alle bestanden in een directory opvragen met:

```
$ ls Plaatje\.*
```

We krijgen dan de bestanden met een willekeurige extensie terug.

Globbing is een functie die door de shell wordt uitgevoerd. De shell vervangt dus eerst de wildcards door wat er op het bestandssysteem staat en voert dat uiteindelijk aan `ls`. Als je quotes om het argument zet dan gaat

`ls` opzoek naar een bestand dat `Plaatje.*` heet. Er verschijnt dan keurig de melding dat dat bestand niet bestaat.

Zo kunnen we ook zoeken op de bestandsnamen waarvan we niet meer zeker weten of we het met een hoofdletter of kleine letter hebben geschreven door gebruikt te maken van het vraagteken:

```
$ ls ?laatje.*
```

Het toeval wil dat we ooit het recept van een vriend hebben opgeschreven over hoe we huzarensalade moeten maken dus kregen we van het laatste commando terug:

```
plaatje.jpg  Plaatje.png Slaatje.txt
```

Als we alleen de bestandsnamen die echt het plaatje bevatten willen vinden dan zullen we gebruik moeten maken van een ander optie die de shell ons biedt, namelijk de blokhaken ([]). In shell-globbing betekenen de blokhaken een reeks. Een reeks kan zijn [1234567890], of [abcdefg], wat trouwens simpeler geschreven kan worden als [0-9] en [a-g], maar een reeks mag ook zo simpel zijn als [pP], dus de hoofdletter P en de kleine letter p.

```
$ ls [Pp]laatje.*
```

en zo krijgen we precies terug wat we willen.

Reeksen in globbing zijn heel handig. Je kan bijvoorbeeld een reeks maken [a-zA-Z0-9] zodat je 3 reeksen combineert en zo alles hebt dat geen leesteken bevat. Als je ook bestandsnamen hebt die een spatie kan bevatten maak je er [\ a-zA-Z-0-9] van. Let op het escape (\) character voor de spatie. Zonder dat character zou de shell aan ls twee argumenten meegeven, want er staat een spatie en spaties (white-space) scheiden argumenten. Dus ls krijgt de opdracht om een list te doen van '[' en van 'a-zA-Z0-9]' en dat is niet wat we willen. We willen dat onze reeks één argument is, vandaar dat we de spatie "escapen" zoals dat heet. Door het escape-character behandelt de shell de spatie niet als scheidingsteken van argumenten maar als onderdeel van de reeks.

3.2 find

Om op de commandline bestanden te zoeken die ergens op het filesysteem staan is er het commando `find`. De syntax van het `find` commando ziet er ongeveer zo uit:

```
find <path> <options> <action>
```

`find` zoekt vanaf het opgegeven `path` naar bestanden die voldoen aan de opgegeven opties en voert daar de opgegeven `action` op uit. De standaard `action` is om de naam van het bestand inclusief het complete pad te printen naar de standaard output. De zoekactie van `find` is als je dat niet limiteert recursief en gaat dus door alle subdirectories. Dat betekent ook dat als je `/` als pad op geeft dat het hele bestandssysteem doorzocht wordt (dat kan even duren).

Op een Linux systeem kunnen er vele directories zijn en is het van belang dat je op een simpele manier bestanden terug kan vinden. Om te zoeken naar bestanden of directories is er het commando `find`. Om een idee te krijgen hoe `find` werkt typen we:

```
$ find ~ -name "Muziek" -print
```

Bij mij op het systeem was het antwoord

```
/home/dennis/LinuxCursus/Muziek
```

Bij jou is dennis natuurlijk weer vervangen door je eigen gebruikersnaam. Print is de standaard functie van `find`, en daarom vonden de programmeurs van `find` dat als je geen opdracht meegeeft dat `find` dan ook print doet. Dus korter kan het zo:

```
$ find ~ -name "Muziek"
```

Die ~is makkelijk als je in je home-directory wilt zoeken, maar wat als je in bijvoorbeeld `/etc/` staat en in die directory wilt zoeken? Daar is ook over nagedacht. We hebben al '..' gezien als een aanduiding voor een lager gelegen directory, maar zo is er ook de '.' als we 'deze'-directory bedoel. En met deze bedoelen we de directory waarin we nu staan. Dus we kunnen ook het volgende doen:

```
$ find . -name "Muziek"
```

En voor wie niets tegen typen heeft mag je natuurlijk ook de hele directory opgeven

```
$ find /home/dennis/ -name "Muziek"
```

Met -name geven we op dat we naar een bestandsnaam zoeken en een bestandsnaam kan ook een directory zijn. Als we onderscheidt willen maken tussen bestanden en directories kan kunnen we een -type meegeven:

```
$ find . -type d -name "Muziek"
```

Zal dezelfde output geven want type d is een directory, maar als we doen

```
$ find . -type f -name "Muziek"
```

dan vinden we niets, want het type f is een regulier bestand en Muziek is een directory. Om dat te testen doen we het volgende

```
$ touch ~/LinuxCursus/Documenten/leeg_bestand.txt
$ find . -type f -name "leeg_bestand.txt"
```

Met `touch` kan je een nieuw leeg bestand aanmaken en dat hebben we gedaan en daarna hebben we `find` naar dat nieuwe bestand laten zoeken. Nu lijkt dit een wat onzinnige actie omdat we al weten waar het bestand is, maar het geeft ons een optie om een andere functie van `find` te demonstreren, namelijk het zoeken naar lege bestanden op het systeem:

```
$ find . -size 0
```

de -size optie geldt alleen voor bestanden, de -empty optie laat naast lege bestanden ook lege directories zien

```
$ find . -empty
```

Het `find` commando kent nog veel meer opties zoals zoeken naar de datum en tijd waarop een bestand is aangemaakt of een moment later of eerder dan een bepaalde datum en tijd. De man-page van `find` documenteert al deze verschillende opties en op Internet is heel veel uitleg te vinden hoe je `find` met al deze opties kan gebruiken. Een laatste functie van `find` willen je nog meegeven. De kan behalve met -print `find` ook andere dingen laten doen dan de gevonden elementen printen, je kan `find` ook vertellen om een actie uit te voeren, zoals het deleten van de gevonden elementen:

```
$ find . -size 0 -delete
```

een `ls` van `LinuxCursus/Documenten/` zal laten zien dat het bestand `leeg_bestand.txt` niet meer bestaat.

Pas wel op, dit kan gevaarlijke situaties opleveren:

```
$ find LinuxCursus/ -empty -delete
```

Dit commando zoekt in de LinuxCursus naar alle directories en bestanden die leeg zijn. Bestanden hadden we niet meer, maar er stonden nog wel twee directories in (Documenten en Muziek), beide directories waren leeg en werden door `find` dan ook keurig verwijderd van het systeem. Toen kwam `find` nog de directory LinuxCursus tegen, en ja die was inmiddels ook leeg, dus heeft `find` die ook weggegooid!

4 Zoeken in bestanden

Soms zou je willen dat je in een bestand kunt zoeken. Natuurlijk kan je met in een tekstverwerker of een editor zoeken in een bestand, maar wat nu

als je niet zeker meer weet in welk bestand het was dat je iets geschreven had. Dat kan zomaar gebeuren als je een boek zoals dit aan het schrijven bent. Dit boek is opgebouwd uit allemaal kleine bestanden die te samen het boek vormen. Op deze manier hou ik de onderwerpen gescheiden en hoef ik niet elke keer te scrollen om bij een ander deel te komen. Ik kan ook twee onderwerpen in twee verschillende terminals te gelijk open hebben staan en zo parallel aan de stukken werken. Dit deel gaat over hoe we in bestanden kunnen zoeken zonder dat we een tekstverwerker hoeven te gebruiken.

Er zijn verschillende commando's die we kunnen gebruiken, de meest gebruikte is denk ik `grep`. Met `grep` kan je door regular expressions te gebruiken zoeken in bestanden. Meer over regular expressions vind je in de volgende sectie. Nu gaan we vooral kijken naar hoe het `grep` commando werkt.

4.1 grep

`grep` is de Global Regular Expression Parser. De naam `grep` vindt zijn oorsprong in een zoekopdracht uit `ed`, een editor: `g/re/p` wat stond voor zoek door de hele tekst (**g**lobal) naar deze **r**egular **e**xpression en druk deze af (**p**rint). `re` werd dan vervangen door een regular expression.

Om te zien hoe `grep` werkt gaan we eerst een paar bestanden met inhoud aanmaken.

```
cd ~
mkdir ZIB
echo "Een commando op Unix \
systemen is vaak klein \
en simpel en doet 1 \
ding goed." > ZIB/kleineCommandos.txt
echo "Door verschillende van deze tools \
te combineren kunnen complexe taken \
volbracht worden." > ZIB/complexeTaken.txt
echo "De documenten voor dit boek \
houden hetzelfde principe aan." > ZIB/boek.txt
echo "Het zijn korte stukken tekst \
die gezamenlijk een compleet boek vormen. \
We noemen dit het \
small-is-beautiful-principe." > ZIB/small_is_beautiful.txt
echo "We kunnen dit op boeken, \
 tekst en software code toepassen." > ZIB/toepassen.txt
```

4.2 Regular Expressions

Regular expressions zijn speciale karakters waarmee je in complexe data-sets kan zoeken. De naam regular expression wordt vaak afgekort tot regexp of regex. Regular expressions worden gebruikt om patronen te zoeken in data. Wordt dit patroon gevonden dan wordt de gehele regel waarin dit patroon voorkomt afgebeeld op het scherm. Je kan zoeken in bestanden of in de output (stdout) van een commando.

De basis van regular expressions is een aantal karakters met een speciale betekenis:

.	Is een willekeurig karakter
?	Is exact 1 karakter
*	Herhaal de voorgaande expressie 0 of meer keren
^	Begin van de regel
$	Einde van de regel
\	Escape; speciale karakters worden als echte karakters behandeld
{}	Zorg dat dat een regular expression een eenheid (groep) vormt

Deze speciale karakters kunnen we gebruiken om regular expressions te maken. We beginnen met de ., ? en *:

```
$ mkdir regex
$ cd regex
$ touch aap.txt
$ touch ap.txt
$ touch pa
$ touch filetxt
$ ls | grep a?*p
$ ls | grep a.*p
```

merk op dat het verschil in output van de twee `ls` commando's zit in het aantal keren dat een letter voor moet komen. De combinatie ?* zegt dat er 1 of meer karakters moeten zijn, terwijl .* zegt dat het 0 of meer karakters moet zijn.

```
$ ls | grep ^a
$ ls | grep a$
```

Met het ^ zoeken we vanaf het begin van de regel, dus de regel moet beginnen met een a. Met de $ zoeken we vanaf het einde, dus de regel moet eindigen met een a.

Als we bestanden willen zien die op .txt eindigen dan willen we dat de punt onderdeel is van onze zoek opdracht en willen we niet dat de punt gezien wordt als een willekeurig karakter. Vergelijk de volgende twee commando's en hun uitkomst:

```
$ ls | grep '.txt'
$ ls | grep '\.txt'
```

Let op de ticks om de regular expression. De ticks zorgen ervoor dat de regular expression bij `grep` terecht komt en niet al door de shell wordt uitgevoerd. De shell is natuurlijk de eerste die de complete commando regel krijgt. Het is de shell die moet beslissen wie welk deel uitvoert. Voor `ls` is dat niet zo moeilijk daar wordt de opdracht aan `ls` overgelaten, maar voor de regular expression bij `grep` ontstaan er twee mogelijkheden. De shell lost zelf al een deel van de regular expression op en geeft wat er over blijft aan `grep` of de shell geeft de complete regular expression aan `grep`. Dat laatste is wat we in dit geval willen. Vergelijk je hiervoor gekregen uitkomsten eens met:

```
$ ls | grep \.txt
```

Wat hopelijk opvalt is dat ondanks de \ er niet gefilterd wordt op .txt maar op alles dat een willekeurig karakter heeft met daarachter txt. Dat komt omdat de shell de \. al matched met de output van `ls`. Dit is dus wel iets om op te letten met regular expressions. Het is altijd veilig om de regex tussen ticks te zetten. Dan weet je zeker dat de regex bij `grep` uitkomt en niet bij de shell.

Soms willen we kunnen aangeven hoe vaak een bepaald karakter achter elkaar voorkomt. Daarvoor hebben we de curly braces (accolades).

```
$ ls | grep -E 'a{2}'
```

Zoek zelf in de man-page van grep op wat -E betekent.

4.3 Zoek en vervang

We kunnen ook regels zoeken in een bestand die aan een bepaald patroon (regex) voldoen en dan het patroon vervangen door iets anders. Een van de meest gebruikte tools daarvoor heet `sed`.

```
$ ls -1
$ ls | sed -e 's/aap/noot/'
```

overal waar aap stond staat nu noot. Op de disk is er niets gewijzigd, we hebben alleen de output van ls aangepast. We hebben de "taak" voor `sed` tussen ticks gezet omdat we zeker willen weten dat deze opdracht bij `sed` terecht komt en dat niet de shell zich ermee gaat bemoeien.

Om iets meer van `sed` te leren gaan we eerst een bestand aanmaken. Gebruik hiervoor `vi`, `vim` of een andere editor.

```
Me stage heeft plaatsgevonden in 2018. De reden dat ik voor dit bedrijf
    gekozen heb is omdat het een overheidsinstantie is en graag wil leren
    hoe is om in zo'n bedrijf te werken, verder wil ik me ontwikkelen
    doormiddel het uitvoeren van wat ik heb geleerd op me school.

Me werkzaamheden was het helpen van mensen met hun probleem binnen en
    buiten het bedrijf, ik zat op de Servicedesk.

Wij krijgen onze technologische aparaten waaronder laptops computers als
    er iets stuk en niet zomaar te repareren valt. We vragen eerst of er
    nog garantie op de product is, zo ja sturen we de produkten naar op
    naar de leverancier. Zo niet dan kijk we er zelf naar.

Ik had een werkende computer die hele tijd vast liep en rare geluide
    maakte. Ik heb zelf een stappenplan gemaakt om te controleren of de
    computer hardware of software problemen had.
    1. Zit de stroom goed op de voeding?
    2. Zijn alle componenten goed aangesloten
    3. Is de product schoon (stofvrij)
    4. Handige Informatie opschrijven van de BIOS

Uiteindelijk was de fout dat er teveel stof in de roosters zat van de
    computer, stof kan computers slecht laten draaien en de lucht
    cirkulatie verminderen van binnen. Waardoor ze kunnen vastlopen.

We gebruiken Topdesk. Dit is een melding registratie systeem. Hierin
    worden incidenten genoteerd en is er een stukje communicatie tussen
    de melder en de oplosser. De melder of een van de servicedesk
    medewerkers maken de melding. Daarna kijkt de oplosser naar de
    melding en stuurt een mail naar de des betreffende persoon. De
    oplosser belt vaak ook naar de melder om de probleem op te lossen en
    meld de melding af na afloop. De melder ontvangt een mail dat de
    melding is afgehandeld.
```

In deze tekst zitten wat taalfouten. Die gaan we zoeken en vervangen. We beginnen met het gebruik van Me aan het begin van de eerste twee alinea's. Dat moet natuurlijk Mijn zijn.

```
$ sed -e 's/^Me/Mijn/'
```

Op het scherm zien we de verbeterde tekst, op disk staat echter nog de oude tekst. We kunnen de verbeterde tekst natuurlijk met het groter dan teken wegschrijven naar disk met een nieuwe bestandsnaam. Makkelijker is het om de verbeterde tekst in het al bestaande bestand aan te passen:

```
$ sed -ie 's/^Me/Mijn/'
```

We zien nu geen output, omdat de tekst op de disk gewijzigd is. Gebruik

`cat` om te zien dat de tekst inderdaad gewijzigd is. Zoek in de man-page op wat de betekenis is van de -i en de -e opties.

Een ander 'me' die fout is is de 'me' voor 'me school' maar de 'me' voor 'me ontwikkelen' is bijvoorbeeld goed. We moeten dus een regular expression schrijven ide alleen matched op 'me school':

```
$ sed -ie 's/me\ school.$/mijn\ school/'
```

Let op de \ voor de spatie. Een spatie is in de shell een scheidingsteken en dat willen we nu niet. We bedoelen een echte spatie, dus moeten we hem escapen.

Index