

# Tekstbestanden en bestandssystemen

Hst 4 en 5 uit het leerboek linux deel 1

Cees Keyer.

LinuxHoorn

7 januari 2006



Amsterdam School of Technology  
Dept. of Electronic Engineering.  
E-technology



Cees Keyer:

Docent op de Hogeschool van Amsterdam Elektrotechniek.

- EMC
- Hoog frequente elektronica.
- Computer Architectuur.
- Communicatie systemen.

Jaren ervaring (15) als systeembeheerder unix (HP-UX 8, 9 en 10)

Mededelingen en andere inge.

# Reguliere expressies



Wat zijn reguliere expressies?

Wordt niet gedefinieerd in het boek.

Een expressie is iets met een uitdrukking.

Een reguliere expressie is een manier om een bepaalde groep tekens te beschrijven of omschrijven.

## Definition

Een reguliere expressie omschrijft een verzameling tekenreeksen (strings) zonder ze allemaal op te noemen. De drie strings Handel, Händel en Haendel kunnen bijvoorbeeld beschreven worden met het patroon "H(a|ä|ae)ndel". Met dank aan wikipedia

# Reguliere expressies 2



De meest eenvoudige reguliere expressie is de volledige tekst. Zoek je bijvoorbeeld het woord *worst* in een file dan kun je het aldus inkloppen.

Example

```
grep worst *
```

# Reguliere expressies 2



De meest eenvoudige reguliere expressie is de volledige tekst. Zoek je bijvoorbeeld het woord *worst* in een file dan kun je het aldus inkloppen.

## Example

```
grep worst *
```

# Reguliere expressies 3



Het gebruik van variabelen. In het linux OS en Unix zijn een aantal standaard tekens afgesproken voor reguliere expressies.

- **^** deze verwijst naar het begin van de regel.
- **\$** Deze verwijst naar het eind van de regel.
- **.** De wild card, deze verwijst naar ieder teken behalve het newline karakter.
- **[ ]** Karakters tussen deze blokhaken worden gezien als een of functie.
- **[ \^ ]** Negeert de tussen de blokhaken gegeven tekens.
- **\ <** Zoekt naar tekens aan het begin van een woord.
- **\ >** aan het eind van een woord.
- **\** is een shell escape, dan wordt een teken niet door de shell uitgevoerd. Voorbeeld  
`find / -name '*.ps' -exec grep bla {} \;`



# Reguliere expressies 3

Het gebruik van variabelen. In het linux OS en Unix zijn een aantal standaard tekens afgesproken voor reguliere expressies.

- $\wedge$  deze verwijst naar het begin van de regel.
- $\$$  Deze verwijst naar het eind van de regel.
- $.$  De wild card, deze verwijst naar ieder teken behalve het newline karakter.
- $[]$  Karakters tussen deze blokhaken worden gezien als een of functie.
- $[\wedge]$  Negeert de tussen de blokhaken gegeven tekens.
- $\backslash <$  Zoekt naar tekens aan het begin van een woord.
- $\backslash >$  aan het eind van een woord.
- $\backslash$  is een shell escape, dan wordt een teken niet door de shell uitgevoerd. Voorbeeld  
`find / -name '*.ps' -exec grep bla {} \;`



# Reguliere expressies 3

Het gebruik van variabelen. In het linux OS en Unix zijn een aantal standaard tekens afgesproken voor reguliere expressies.

- $\wedge$  deze verwijst naar het begin van de regel.
- $\$$  Deze verwijst naar het eind van de regel.
- $\cdot$  De wild card, deze verwijst naar ieder teken behalve het newline karakter.
- $[ ]$  Karakters tussen deze blokhaken worden gezien als een of functie.
- $[ \wedge ]$  Negeert de tussen de blokhaken gegeven tekens.
- $\backslash <$  Zoekt naar tekens aan het begin van een woord.
- $\backslash >$  aan het eind van een woord.
- $\backslash$  is een shell escape, dan wordt een teken niet door de shell uitgevoerd. Voorbeeld  
`find / -name '*.ps' -exec grep bla {} \;`



# Reguliere expressies 3

Het gebruik van variabelen. In het linux OS en Unix zijn een aantal standaard tekens afgesproken voor reguliere expressies.

- $\wedge$  deze verwijst naar het begin van de regel.
- $\$$  Deze verwijst naar het eind van de regel.
- $.$  De wild card, deze verwijst naar ieder teken behalve het newline karakter.
- $[ ]$  Karakters tussen deze blokhaken worden gezien als een of functie.
- $[ \wedge ]$  Negeert de tussen de blokhaken gegeven tekens.
- $\backslash <$  Zoekt naar tekens aan het begin van een woord.
- $\backslash >$  aan het eind van een woord.
- $\backslash$  is een shell escape, dan wordt een teken niet door de shell uitgevoerd. Voorbeeld  
`find / -name '*.ps' -exec grep bla {} \;`



# Reguliere expressies 3

Het gebruik van variabelen. In het linux OS en Unix zijn een aantal standaard tekens afgesproken voor reguliere expressies.

- $\wedge$  deze verwijst naar het begin van de regel.
- $\$$  Deze verwijst naar het eind van de regel.
- $.$  De wild card, deze verwijst naar ieder teken behalve het newline karakter.
- $[ ]$  Karakters tussen deze blokhaken worden gezien als een of functie.
- $[ \wedge ]$  Negeert de tussen de blokhaken gegeven tekens.
- $\backslash <$  Zoekt naar tekens aan het begin van een woord.
- $\backslash >$  aan het eind van een woord.
- $\backslash$  is een shell escape, dan wordt een teken niet door de shell uitgevoerd. Voorbeeld  
`find / -name '*.ps' -exec grep bla {} \;`



# Reguliere expressies 3

Het gebruik van variabelen. In het linux OS en Unix zijn een aantal standaard tekens afgesproken voor reguliere expressies.

- $\wedge$  deze verwijst naar het begin van de regel.
- $\$$  Deze verwijst naar het eind van de regel.
- $.$  De wild card, deze verwijst naar ieder teken behalve het newline karakter.
- $[ ]$  Karakters tussen deze blokhaken worden gezien als een of functie.
- $[ \wedge ]$  Negeert de tussen de blokhaken gegeven tekens.
- $\backslash <$  Zoekt naar tekens aan het begin van een woord.
- $\backslash >$  aan het eind van een woord.
- $\backslash$  is een shell escape, dan wordt een teken niet door de shell uitgevoerd. Voorbeeld  
`find / -name '*.ps' -exec grep bla {} \;`



# Reguliere expressies 3

Het gebruik van variabelen. In het linux OS en Unix zijn een aantal standaard tekens afgesproken voor reguliere expressies.

- `^` deze verwijst naar het begin van de regel.
- `$` Deze verwijst naar het eind van de regel.
- `.` De wild card, deze verwijst naar ieder teken behalve het newline karakter.
- `[]` Karakters tussen deze blokhaken worden gezien als een of functie.
- `[^]` Negeert de tussen de blokhaken gegeven tekens.
- `\<` Zoekt naar tekens aan het begin van een woord.
- `\>` aan het eind van een woord.
- `\` is een shell escape, dan wordt een teken niet door de shell uitgevoerd. Voorbeeld  
`find / -name '*.ps' -exec grep bla {} \;`



# Reguliere expressies 4

De vorige 7 sheets met operatoren zijn bruikbaar tijdens zoektochten naar teken reeksen (strings) Er zijn ook herhalings operatoren.

Bij het veel gebruikte grep (RTFM) moet je wel de reguliere expressie tussen aanhalingstekens plaatsen (single quotes) als bescherming van de regex tegen de shell.

## Example

Zoek de string \$HOME in een file. `grep '$HOME' <filenaam>`

Probeer dit zelf en kijk goed naar het verschil tussen:

`grep '$HOME' <filenaam>` EN

`grep '$HOME' <filenaam>`

`grep -E $HOME <filenaam>`

Ziek ook <http://analyser.oli.tudelft.nl/regex/index.html.nl> voor meer info.

# Reguliere expressies 5



- \* het voorgaande teken maar meerdere malen 1 maal of helemaal niet voorkomen. Het maakt dus niet uit.
- ? Het voorgaande teken of de reguliere expressie hoeft niet voor te komen maar komt maximaal één keer voor.  
grep tekst? bla levert op:  
teks tekst teksts



# Reguliere expressies 6

- + Het voorgaande teken of regex moet minstens één maal voorkomen.
- $\{n\}$  het voorgaande teken of regex moet precies n keer achter elkaar voorkomen. De \ zijn shell escapes.
- $\{n, \}$  Het voorgaande teken of regex komt minimaal n keer voor.
- $\{, m\}$  Het voorgaande teken of regex komt hooguit m keer voor maar nul keer kan dus ook.
- $\{n, m\}$  Het voorgaande teken of regex komt minimaal n en maximaal m keer voor.
- $\backslash(\backslash)$  Bewaar het teken tussen de haakjes in holding space. Maximaal 9 tekens in de holdingspace hier naar kun je verwijzen met  $\backslash6$  voor plaats 6 in de holding space.
- `regex|regex` Zoek naar ofwel teken voor ofwel teken na het | teken.



Het combinatie van find en grep zijn uitermate geschikt voor het zoeken van tekst in files. Het advies is wel om altijd even naar de manual te kijken, er zijn nog al wat verschillende implementaties.

**grep** Met grep kun je naar strings zoeken in een file.

**find** Met find kun je naar files directories etc zoeken en daar commando's op los laten. Vergeet nooit de "ge-escape" ; als je de optie -exec gebruikt.



# Stdout Stdin Stderr

Onderlinux zijn er 3 stromen van data te onderscheiden. Staat niet in het boek of ik heb niet goed gezocht.

**stdin** Standard In is een data stroom die meestal van je keyboard komt. Je kunt de output van een commando ook redirecten naar input van een ander programma. Dit doe je met het | teken of als je het in een file wilt duwen met het > teken.

**stdout** Dit is meestal de uitvoer die op je scherm wordt getoond. Maar kan ook als invoer dienen voor een andere prog, dus stdout van 1 prog wordt stdin van een ander prog. zie ook voorgaand punt.

**stderr** standard Error Hier worden fout meldingen op uitgespuugd.

Zie ook paragraaf 2.4.2 van het boek en nu is ook duidelijk waarom het redirection wordt genoemd. Het omleiden van stdout naar stdin van een ander prog.....

# sort cut uniq cat



- sort** Met dit commando kun je de inhoud van een file sorteren en de gesorteerde uitvoer wordt op **stdout** gepresenteerd. Zie manual voor de opties.
- cut** Stuurt bepaalde velden uit een file naar **stdout**
- uniq** met dit commando kun je de inhoud van een file uniek maken, dat wil zeggen als er meerdere dezelfde regels in een file zitten dan wordt er slechts 1 overgelaten.
- cat** Dit commando, een afkorting van concatenate wat globaal aan elkaar rijgen betekend, plakt files achter elkaar. Werkt ook op 1 file. Stdout is de normale uitvoer. Handig om te gebruiken is I/O redirection.



# tr, nl, od, split, join

- tr** TR is de afkorting van translate, vervangt karakters voor andere of verwijderd deze. voorbeeld `tr -d * <bla >bla1`
- nl** Laat de inhoud van een bestand zien op stdout voorzien van regelnummers.
- od** Octal Dump. Laat de inhoud van een bestand octaal zien op stdout. Let op devices zijn onder linux ook bestanden, dus de inhoud van een beschadigde disk kun je vaak nog octaal ophalen en in een file duwen.
- split** Verdeeld een file in kleinere stukken. output wordt opeenvolgend genummerd.
- join** Join is niet split maar voegt dus files samen.

# Sed en (g)awk



Bij de volgende tools worden heel vaak reguliere expressies gebruikt. Echter wordt tegenwoordig steeds vaker gegrepen naar perl of zijn soortgenoten.

**SED** Stream editor, zoals de naam al suggereert is dit een editor voor een data stroom. Dus laat de inhoud zien van een file duw dat door sed, die doet er iets mee en stop het in een andere file. `cat file1 | sed -opties >file2` dus.

**(g)awk** Awk is ooit bedacht door Aho, Weinberger, & Kernighan van daar de naam awk. het is een interpreteerbare programmeer taal. Het is nuttig als opstapje voor -de programeer taal C die bedacht is door Kernighan samen met Ritchie die weer de grondslag hebben gelegd voor Unix..... Een goede introductie voor awk is te vinden op: <http://www.vectorsite.net/tsawk.html>

# sed en gawk 2



Sed en awk zijn niet te leren door het luisteren naar een verhaal, dus moet je het zelf allemaal doen. Kortom ga er mee aan de gang. De substitutie opdrachten die je in sed gebruikt kun je ook in vi gebruiken. In vi doe je dit door in commando mode de `:` in te kloppen.

## Definition

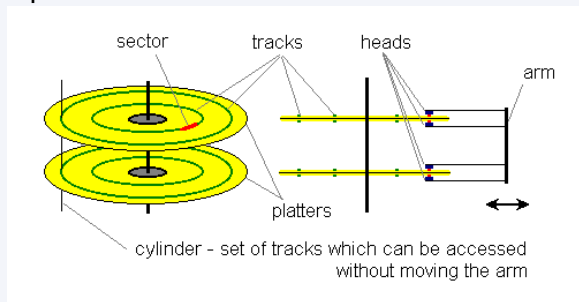
De manier waarop data op een data drager wordt georganiseerd is een bestandstelsysteem. Dit is dus niet beperkt tot een hard disk.

Voorbeelden van datadragers.

- Floppy disk (flopperschijf in het vlaams.)
- CD en DVD
- magtape (real en cartridge)
- ramdisk
- USB stick en zijn soortgenoten.



## Opbouw harddisk:



Ieder os heeft zijn eigen HD bestandstelsysteem:

**dos** File Allocation Table FAT 16 k en 32 k blokken.

**windows** HPFS (xp) NTFS (nt en gejat van os2) FAT W95 en W98

**Linux** reiser fs, swap, fat, etc

Linux maakt gebruik van een inode georienteerd filesystem.

### Definition

Een inode is een datastructuur waarin de belangrijke gegevens van een block data zijn opgeslagen. Zoals owner, group access rechten en vooral waar het op disk staat.



De inode's worden opgeslagen in een tabel dat aan het begin van de harddisk staat. Deze tabel wordt om de zoveel sectoren herhaald om te voorkomen dat er te lang naar gezocht moet worden.



- `ls -i <dir naam>` Laat de inhoud aan files zien van de directory `<dir naam>` met de bijbehorende i-nodes.

## Example

```
cees@edradour:~/documents/prive/ochtend> ls -i 311580
examples 311493 ochtend.dvi 311492 ochtend.out 311496
ochtend.snm 311476 ochtend.aux 311494 ochtend.nav 311569
ochtend..pdf 311495 ochtend.toc
```



Twee typen een symbolic en de echte link.

**symbolic** Dit is een koppeling naar de naam van het bestand. Werkt over media heen. Vervalt na het verwijderen van het origineel.

**hard** Is koppeling naar inode van bestand. beperkt tot 1 medium(partitie)

Hoe maak je een symbolic link.

In `-s /etc/passwd /tmp/bla`

Hoe maak je een hard link.

In `/etc/passwd /tmp/bla`

# harddisk benaderbaar maken.



Het engelse woord to mount betekend zoveel als iets bestijgen.

Begrippen:

- **mountpoint** De directory in het root filesystem (of root zelf) waar een disk of partitie van een disk gemount kan worden.
- **/etc/fstab** de file met alle te mounten disks, wordt gelezen tijdens het opstarten. (Dit impliceert dat /etc in de root dir en op de root partitie moet staan.
- **device files** De block devices zoals die in de /dev staan. Dit zijn bijzondere files waarachter de kernel zit die de toegang tot het device verleent. Dit kan al dan niet raw data zijn.

Met mount kun je een partitie aan een dir plakken. Met umount weer los maken. vb. `mount /dev/fd0 /media/floppy`  
`/media/floppy` moet wel eerst bestaan. `umount /media/floppy` maakt de floppy weer "los" ui het os. Veel voorkomende foutmelding `devices busy..` spreekt voor zich.



Er zijn twee commando's die de gebruiker niet zo vaak zal gebruiken.

fsck FileSystem CHeck. Dit commando voert een filesystem check uit om te kijken of alle inode tabellen etc nog kloppen. ext 2 en ext3 bestandssystemen worden automatisch na zoveel mounts gechecked.

mkfs komt van Make File System en dit spreekt voorzich. Vergelijk het formateren onder windows of dos.