# Linux commands

Ten behoeve van beeldvorming m.b.t. shell e.d.

10-mrt-2014

# Pipe & Filter

- `Pump -pipe-> Filter1 -pipe-> Filter2 -pipe-> Sink`

# Pipe & Filter

- `Pump -pipe-> Filter1 -pipe-> Filter2 -pipe-> Sink`
- De pump is de event generator, zou je kunnen zien als eerste filter.

# Pipe & Filter

- `Pump -pipe-> Filter1 -pipe-> Filter2 -pipe-> Sink`
- De pump is de event generator, zou je kunnen zien als eerste filter.
- Eindpunt van de stream komt in 'sink' terecht (i.e. terminal of file)

# Pipe & Filter

- `Pump -pipe-> Filter1 -pipe-> Filter2 -pipe-> Sink`
- De pump is de event generator, zou je kunnen zien als eerste filter.
- Eindpunt van de stream komt in 'sink' terecht (i.e. terminal of file)
- In linux typisch herkenbaar als:

# Pipe & Filter

- `Pump -pipe-> Filter1 -pipe-> Filter2 -pipe-> Sink`
- De pump is de event generator, zou je kunnen zien als eerste filter.
- Eindpunt van de stream komt in 'sink' terecht (i.e. terminal of file)
- In linux typisch herkenbaar als:
- `shell$ ls | sort | uniq`
- `shell$ cat | sort | uniq`

# Pipe & Filter

- `Pump -pipe-> Filter1 -pipe-> Filter2 -pipe-> Sink`
- De pump is de event generator, zou je kunnen zien als eerste filter.
- Eindpunt van de stream komt in 'sink' terecht (i.e. terminal of file)
- In linux typisch herkenbaar als:
- `shell$ ls | sort | uniq`
- `shell$ cat | sort | uniq`
- `shell$ cat < input.txt | sort | uniq`

# Pipe & Filter

- Door die pipes gaat een stream. Normaal gesproken drie:

```
$ ls -althr /dev/std*
lrwxrwxrwx 1 root root 15 mrt  3 10:30 /dev/stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root 15 mrt  3 10:30 /dev/stdout -> /proc/self/fd/1
lrwxrwxrwx 1 root root 15 mrt  3 10:30 /dev/stderr -> /proc/self/fd/2
```

# Pipe & Filter

- Door die pipes gaat een stream. Normaal gesproken drie:

```
$ ls -althr /dev/std*
lrwxrwxrwx 1 root root 15 mrt  3 10:30 /dev/stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root 15 mrt  3 10:30 /dev/stdout -> /proc/self/fd/1
lrwxrwxrwx 1 root root 15 mrt  3 10:30 /dev/stderr -> /proc/self/fd/2
```

- Elk proces heeft zijn eigen stdin/stdout/stderr stream

# Pipe & Filter

- Door die pipes gaat een stream. Normaal gesproken drie:

```
$ ls -althr /dev/std*
lrwxrwxrwx 1 root root 15 mrt  3 10:30 /dev/stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root 15 mrt  3 10:30 /dev/stdout -> /proc/self/fd/1
lrwxrwxrwx 1 root root 15 mrt  3 10:30 /dev/stderr -> /proc/self/fd/2
```

- Elk proces heeft zijn eigen stdin/stdout/stderr stream
- Een filter krijgt een stream binnen via stdin, en output via stdout.

  Het idee is dat stderr gebruikt wordt voor errors en gaan altijd *direct*

  naar de "Sink" .

# Voorbeeld filter snorkel(.cpp)

```
01.  #include <iostream>
02.  #include <string>
03.
04.  using namespace std;
05.
06.  int main()
07.  {
08.      string input;
09.      while (cin >> input) {
10.          if (input == "spin") {
11.              cerr << "ieuw!" << endl;
12.          }
13.          cout << "Lol een " << input << " ;-)" << endl;
14.      }
15.  }
```

# Voorbeeld filter snorkel(.cpp)

```
01.     $ printf "appel\nspin\nbanaan\n" | snorkel
```

# Voorbeeld filter snorkel(.cpp)

```
$ printf "appel\nspin\nbanaan\n" | snorkel
Lol een appel ;-)
ieuw!
Lol een spin ;-)
Lol een banaan ;-)
```

# Voorbeeld filter snorkel(.cpp)

```
01.     $ printf "appel\nspin\nbanaan\n" | snorkel
02.     Lol een appel ;-)
03.     ieuw!
04.     Lol een spin ;-)
05.     Lol een banaan ;-)

06.     $ printf "appel\nspin\nbanaan\n" | snorkel | cat
```

# Voorbeeld filter snorkel(.cpp)

```
01.     $ printf "appel\nspin\nbanaan\n" | snorkel
02.     Lol een appel ;-)
03.     ieuw!
04.     Lol een spin ;-)
05.     Lol een banaan ;-)

06.     $ printf "appel\nspin\nbanaan\n" | snorkel | cat
07.     ieuw!
08.     Lol een appel ;-)
09.     Lol een spin ;-)
10.     Lol een banaan ;-)
```

# Voorbeeld filter snorkel(.cpp)

```
$ printf "appel\nspin\nbanaan\n" | snorkel | print
```

# Voorbeeld filter snorkel(.cpp)

```
01.     $ printf "appel\nspin\nbanaan\n" | snorkel | print
02.     ieuw!
03.
```

print leest niets van stdin en print print zelf alleen een lege string + newline.

[Edit: als je ksh gebruikt, you're gonna have a bad time (blijkbaar)]

# Voorbeeld filter snorkel(.cpp)

```
01.     $ printf "appel\nspin\nbanaan\n" | snorkel | print
02.     ieuw!
03.
```

print leest niets van stdin en print print zelf alleen een lege string + newline.

[Edit: als je ksh gebruikt, you're gonna have a bad time (blijkbaar)]

```
$ printf "appel\nspin\nbanaan\n" | snorkel 2>&1 | cat
```

# Voorbeeld filter snorkel(.cpp)

```
$ printf "appel\nspin\nbanaan\n" | snorkel | print
ieuw!

```

print leest niets van stdin en print print zelf alleen een lege string + newline.

[Edit: als je ksh gebruikt, you're gonna have a bad time (blijkbaar)]

```
$ printf "appel\nspin\nbanaan\n" | snorkel 2>&1 | cat
Lol een appel ;-)
ieuw!
Lol een spin ;-)
Lol een banaan ;-)
```

# Voorbeeld filter snorkel(.cpp)

```
$ printf "appel\nspin\nbanaan\n" | snorkel 2>&1 >> combined.log
```

# Voorbeeld filter snorkel(.cpp)

```
$ printf "appel\nspin\nbanaan\n" | snorkel >> combined.log 2>&1
```

# Voorbeeld filter snorkel(.cpp)

```
01.     $ printf "appel\nspin\nbanaan\n" | snorkel >> combined.log 2>&1
02.
03.     $ printf "appel\nspin\nbanaan\n" | snorkel 1>>stdout.log 2>>stderr.log
```

# Voorbeeld filter snorkel(.cpp)

```
01.     $ printf "appel\nspin\nbanaan\n" | snorkel >> combined.log 2>&1
02.
03.     $ printf "appel\nspin\nbanaan\n" | snorkel 1>>stdout.log 2>>stderr.log
04.
05.     >> is append
06.      > is truncate en append
```

# Voorbeeld filter snorkel(.cpp)

```
01.     $ printf "appel\nspin\nbanaan\n" | snorkel >> combined.log 2>&1
02.
03.     $ printf "appel\nspin\nbanaan\n" | snorkel 1>>stdout.log 2>>stderr.log
04.
05.     >> is append
06.      > is truncate en append
07.
08.     $ printf "appel\nspin\nbanaan\n" | snorkel 2>/dev/null
```

# Voorbeeld filter snorkel(.cpp)

```
01.     $ printf "appel\nspin\nbanaan\n" | snorkel >> combined.log 2>&1
02.
03.     $ printf "appel\nspin\nbanaan\n" | snorkel 1>>stdout.log 2>>stderr.log
04.
05.     >> is append
06.      > is truncate en append
07.
08.     $ printf "appel\nspin\nbanaan\n" | snorkel 2>/dev/null
09.     Lol een appel ;-)
10.     Lol een spin ;-)
11.     Lol een banaan ;-)
```

# Voorbeeld filter snorkel(.cpp)

```
01.     $ printf "appel\nspin\nbanaan\n" | snorkel >> combined.log 2>&1
02.
03.     $ printf "appel\nspin\nbanaan\n" | snorkel 1>>stdout.log 2>>stderr.log
04.
05.     >> is append
06.      > is truncate en append
07.
08.     $ printf "appel\nspin\nbanaan\n" | snorkel 2>/dev/null
09.     Lol een appel ;-)
10.     Lol een spin ;-)
11.     Lol een banaan ;-)
12.     $ snorkel < input.txt
```

# Voorbeeld filter snorkel(.cpp)

```
01.     $ printf "appel\nspin\nbanaan\n" | snorkel >> combined.log 2>&1
02.
03.     $ printf "appel\nspin\nbanaan\n" | snorkel 1>>stdout.log 2>>stderr.log
04.
05.     >> is append
06.      > is truncate en append
07.
08.     $ printf "appel\nspin\nbanaan\n" | snorkel 2>/dev/null
09.     Lol een appel ;-)
10.     Lol een spin ;-)
11.     Lol een banaan ;-)
12.     $ snorkel << __TEXT__
13.     appel
14.     spin
15.     banaan
16.     __TEXT__
```

# Filters

- Een filter kan een compiled programma zijn zoals grep, etc. :)
- Een script eerste regel geeft aan wat voor script, i.e.: `#!/bin/bash`
- Kan ook een functie in je shell zijn.

# Filters

Voorbeeld uit mijn .kshrc:

```
01.     function uniq_with_memory
02.     {
03.         typeset -A lines
04.         while read line; do
05.             if [[ ! ${lines[$line]} ]]; then
06.                 echo $line;
07.                 lines[$line]=true
08.             fi
09.         done < /dev/stdin
10.     }
```

# Filters

Voorbeeld uit mijn .kshrc:

```
01.     function uniq_with_memory
02.     {
03.         typeset -A lines
04.         while read line; do
05.             if [[ ! ${lines[$line]} ]]; then
06.                 echo $line;
07.                 lines[$line]=true
08.             fi
09.         done < /dev/stdin
10.     }
01.     printf "a\nb\na\n" | uniq
02.     a
03.     b
04.     a
```

# Filters

Voorbeeld uit mijn .kshrc:

```
01.     function uniq_with_memory
02.     {
03.         typeset -A lines
04.         while read line; do
05.             if [[ ! ${lines[$line]} ]]; then
06.                 echo $line;
07.                 lines[$line]=true
08.             fi
09.         done < /dev/stdin
10.     }
01.     printf "a\nb\na\n" | uniq_with_memory
02.     a
03.     b
```

# Filters

```
01.     function prefix
02.     {
03.         while read line; do
04.             print "$1$line";
05.         done < /dev/stdin
06.     }
```

# Filters

```
01.     function prefix
02.     {
03.         while read line; do
04.             print "$1$line";
05.         done < /dev/stdin
06.     }
01.     $ (smashbattle -s "TRAINING DOJO" 1100 "TEST SERV" 2>&1 | prefix "server: " ) &
02.     $ sleep 2
03.     $ (smashbattle -c localhost 1100 2>&1 |prefix "client: " )&
```

# Filters

```
01.     function prefix
02.     {
03.         while read line; do
04.             print "$1$line";
05.         done < /dev/stdin
06.     }
01.     $ (smashbattle -s "TRAINING DOJO" 1100 "TEST SERV" 2>&1 | prefix "server: " ) &
02.     $ sleep 2
03.     $ (smashbattle -c localhost 1100 2>&1 |prefix "client: " )&
04.     ...
05.     client: CONSOLE CONNECTING TO localhost:1100...
06.     client: CONSOLE CONNECTION SUCCESFUL
07.     server: INFO New client connected with id: 0
08.     server:
09.     client: CONSOLE Send packet of type 1 through UDP with seq 0
10.     client: CONSOLE received packet of type: 0x17
11.     client: CONSOLE Acknowledging communication token from server
12.     ...
```

# Nieuwe "syntax" van vorige slide

- ( ) is een subshell.
- & = run in background.

Voorbeeld je hebt een commando `ddos meetup.com` wat je wilt uitvoeren op een remote machine.

# ddos meetup.com

Wellicht wil je voor het ddos commando dit uitvoeren:

```
01.     $ (sleep 300; killall -9 ddos) &
02.     [3] 26740
```

# ddos meetup.com

Wellicht wil je voor het ddos commando dit uitvoeren:

```
01.     $ (sleep 300; killall -9 ddos) &
02.     [3] 26740
03.     $ ddos meetup.com
04.     packetting meetup.com.......
```

# ddos meetup.com

Wellicht wil je voor het ddos commando dit uitvoeren:

```
01.     $ (sleep 300; killall -9 ddos) &
02.     [3] 26740
03.     laat je die haken weg, gaat je shell eerst 300 seconden slapen
04.     en spawned dan de killall in de background.
05.     $ ddos meetup.com
06.     packetting meetup.com.......
```

# ddos meetup.com / jobs

```
$ jobs
```

# ddos meetup.com / jobs

```
01.     $ jobs
02.     [3] +  Running                 (sleep 300; killall -9 ddos) &
03.     [2] - Stopped                  vim snorkel.cpp
04.     [1]   Stopped                  vim test_servs.ksh
```

# ddos meetup.com / jobs

```
01.     $ jobs
02.     [3] +  Running                 (sleep 300; killall -9 ddos) &
03.     [2] - Stopped                  vim snorkel.cpp
04.     [1]   Stopped                  vim test_servs.ksh
05.     $ kill %3
```

# ddos meetup.com / jobs

```
01.     $ jobs
02.     [3] +  Running                 (sleep 300; killall -9 ddos) &
03.     [2] - Stopped                  vim snorkel.cpp
04.     [1]   Stopped                  vim test_servs.ksh
05.     $ kill %3
06.     [3] + Terminated               (sleep 300; killall -9 ddos) &
07.     sleep is niet een executable maar een commando van de shell,
08.     zal dus niet verschijnen in ps output
```

# ddos meetup.com / jobs

```
01.     $ jobs
02.     [3] +  Running                 (sleep 300; killall -9 ddos) &
03.     [2] - Stopped                  vim snorkel.cpp
04.     [1]   Stopped                  vim test_servs.ksh
05.     $ kill %3
06.     [3] + Terminated               (sleep 300; killall -9 ddos) &
07.     sleep is niet een executable maar een commando van de shell,
08.     zal dus niet verschijnen in ps output
```

Echt een subshell:

```
        $ pwd
```

# ddos meetup.com / jobs

```
01.  $ jobs
02.  [3] +  Running                 (sleep 300; killall -9 ddos) &
03.  [2] - Stopped                  vim snorkel.cpp
04.  [1]   Stopped                  vim test_servs.ksh
05.  $ kill %3
06.  [3] + Terminated               (sleep 300; killall -9 ddos) &
07.  sleep is niet een executable maar een commando van de shell,
08.  zal dus niet verschijnen in ps output
```

Echt een subshell:

```
01.  $ pwd
02.  /home/trigen
```

# ddos meetup.com / jobs

```
01.  $ jobs
02.  [3] +  Running                 (sleep 300; killall -9 ddos) &
03.  [2] - Stopped                  vim snorkel.cpp
04.  [1]   Stopped                  vim test_servs.ksh
05.  $ kill %3
06.  [3] + Terminated               (sleep 300; killall -9 ddos) &
07.  sleep is niet een executable maar een commando van de shell,
08.  zal dus niet verschijnen in ps output
```

Echt een subshell:

```
01.  $ pwd
02.  /home/trigen
03.  $ (cd /usr/local/src; pwd)
```

# ddos meetup.com / jobs

```
01.  $ jobs
02.  [3] +  Running                 (sleep 300; killall -9 ddos) &
03.  [2] - Stopped                  vim snorkel.cpp
04.  [1]   Stopped                  vim test_servs.ksh
05.  $ kill %3
06.  [3] + Terminated               (sleep 300; killall -9 ddos) &
07.  sleep is niet een executable maar een commando van de shell,
08.  zal dus niet verschijnen in ps output
```

Echt een subshell:

```
01.  $ pwd
02.  /home/trigen
03.  $ (cd /usr/local/src; pwd)
04.  /usr/local/src
```

# ddos meetup.com / jobs

```
01.  $ jobs
02.  [3] +  Running                 (sleep 300; killall -9 ddos) &
03.  [2] - Stopped                  vim snorkel.cpp
04.  [1]   Stopped                  vim test_servs.ksh
05.  $ kill %3
06.  [3] + Terminated               (sleep 300; killall -9 ddos) &
07.  sleep is niet een executable maar een commando van de shell,
08.  zal dus niet verschijnen in ps output
```

Echt een subshell:

```
01.  $ pwd
02.  /home/trigen
03.  $ (cd /usr/local/src; pwd)
04.  /usr/local/src
05.  $ pwd
```

# ddos meetup.com / jobs

```
01.  $ jobs
02.  [3] +  Running                 (sleep 300; killall -9 ddos) &
03.  [2] - Stopped                  vim snorkel.cpp
04.  [1]   Stopped                  vim test_servs.ksh
05.  $ kill %3
06.  [3] + Terminated               (sleep 300; killall -9 ddos) &
07.  sleep is niet een executable maar een commando van de shell,
08.  zal dus niet verschijnen in ps output
```

Echt een subshell:

```
01.  $ pwd
02.  /home/trigen
03.  $ (cd /usr/local/src; pwd)
04.  /usr/local/src
05.  $ pwd
06.  /home/trigen
```

# jobs

```
$ jobs
```

# jobs

```
01.     $ jobs
02.     [3] +  Running                 (sleep 300; killall -9 ddos) &
03.     [2] - Stopped                  vim snorkel.cpp
04.     [1]   Stopped                  vim test_servs.ksh
```

# jobs

```
01.     $ jobs
02.     [3] +  Running                 (sleep 300; killall -9 ddos) &
03.     [2] - Stopped                  vim snorkel.cpp
04.     [1]   Stopped                  vim test_servs.ksh
05.     $ fg %3
```

# jobs

```
01.     $ jobs
02.     [3] +  Running                 (sleep 300; killall -9 ddos) &
03.     [2] - Stopped                  vim snorkel.cpp
04.     [1]   Stopped                  vim test_servs.ksh
05.     $ fg %3
06.     (sleep 300; killall -9 ddos)
```

# jobs

```
01.  $ jobs
02.  [3] +  Running                 (sleep 300; killall -9 ddos) &
03.  [2] - Stopped                  vim snorkel.cpp
04.  [1]   Stopped                  vim test_servs.ksh
05.  $ fg %3
06.  (sleep 300; killall -9 ddos)
07.  ^C
```

# jobs

```
01.  $ jobs
02.  [3] +  Running                 (sleep 300; killall -9 ddos) &
03.  [2] - Stopped                  vim snorkel.cpp
04.  [1]   Stopped                  vim test_servs.ksh
05.  $ fg %3
06.  (sleep 300; killall -9 ddos)
07.  ^C
08.  $ jobs
```

# jobs

```
01.  $ jobs
02.  [3] +  Running                 (sleep 300; killall -9 ddos) &
03.  [2] - Stopped                  vim snorkel.cpp
04.  [1]   Stopped                  vim test_servs.ksh
05.  $ fg %3
06.  (sleep 300; killall -9 ddos)
07.  ^C
08.  $ jobs
09.  [2] + Stopped                  vim snorkel.cpp
10.  [1] - Stopped                  vim test_servs.ksh
```

# jobs

Nog een voorbeeld:

```
$ tail -f /var/log/syslog
```

# jobs

Nog een voorbeeld:

```
01.     $ tail -f /var/log/syslog
02.     Mar  3 22:35:26 Firefly21 whoopsie[1386]: online
03.     Mar  3 22:36:43  whoopsie[1386]: last message repeated 2 times
04.     (...)
```

# jobs

Nog een voorbeeld:

```
01.     $ tail -f /var/log/syslog
02.     Mar  3 22:35:26 Firefly21 whoopsie[1386]: online
03.     Mar  3 22:36:43  whoopsie[1386]: last message repeated 2 times
04.     (...)
05.     ^Z
06.     [3] + Stopped                 tail -f /var/log/syslog
```

# jobs

Nog een voorbeeld:

```
01.     $ tail -f /var/log/syslog
02.     Mar  3 22:35:26 Firefly21 whoopsie[1386]: online
03.     Mar  3 22:36:43  whoopsie[1386]: last message repeated 2 times
04.     (...)
05.     ^Z
06.     [3] + Stopped                 tail -f /var/log/syslog
07.     $ jobs
```

# jobs

Nog een voorbeeld:

```
01.     $ tail -f /var/log/syslog
02.     Mar  3 22:35:26 Firefly21 whoopsie[1386]: online
03.     Mar  3 22:36:43  whoopsie[1386]: last message repeated 2 times
04.     (...)
05.     ^Z
06.     [3] + Stopped                 tail -f /var/log/syslog
07.     $ jobs
08.     [3] + Stopped                 tail -f /var/log/syslog
09.     [2] - Stopped                 vim snorkel.cpp
10.     [1]   Stopped                 vim test_servs.ksh
```

# jobs

Nog een voorbeeld:

```
01.     $ tail -f /var/log/syslog
02.     Mar  3 22:35:26 Firefly21 whoopsie[1386]: online
03.     Mar  3 22:36:43  whoopsie[1386]: last message repeated 2 times
04.     (...)
05.     ^Z
06.     [3] + Stopped                 tail -f /var/log/syslog
07.     $ jobs
08.     [3] + Stopped                 tail -f /var/log/syslog
09.     [2] - Stopped                 vim snorkel.cpp
10.     [1]   Stopped                 vim test_servs.ksh
11.     $ bg %3
```

# jobs

Nog een voorbeeld:

```
01.     $ tail -f /var/log/syslog
02.     Mar  3 22:35:26 Firefly21 whoopsie[1386]: online
03.     Mar  3 22:36:43  whoopsie[1386]: last message repeated 2 times
04.     (...)
05.     ^Z
06.     [3] + Stopped                 tail -f /var/log/syslog
07.     $ jobs
08.     [3] + Stopped                 tail -f /var/log/syslog
09.     [2] - Stopped                 vim snorkel.cpp
10.     [1]   Stopped                 vim test_servs.ksh
11.     $ bg %3
12.     [3] tail -f /var/log/syslog&
```

# jobs

Nog een voorbeeld:

```
01.     $ tail -f /var/log/syslog
02.     Mar  3 22:35:26 Firefly21 whoopsie[1386]: online
03.     Mar  3 22:36:43  whoopsie[1386]: last message repeated 2 times
04.     (...)
05.     ^Z
06.     [3] + Stopped                 tail -f /var/log/syslog
07.     $ jobs
08.     [3] + Stopped                 tail -f /var/log/syslog
09.     [2] - Stopped                 vim snorkel.cpp
10.     [1]   Stopped                 vim test_servs.ksh
11.     $ bg %3
12.     [3] tail -f /var/log/syslog&
13.     $
```

# jobs

Nog een voorbeeld:

```
01.  $ tail -f /var/log/syslog
02.  Mar  3 22:35:26 Firefly21 whoopsie[1386]: online
03.  Mar  3 22:36:43  whoopsie[1386]: last message repeated 2 times
04.  (...)
05.  ^Z
06.  [3] + Stopped                 tail -f /var/log/syslog
07.  $ jobs
08.  [3] + Stopped                 tail -f /var/log/syslog
09.  [2] - Stopped                 vim snorkel.cpp
10.  [1]   Stopped                 vim test_servs.ksh
11.  $ bg %3
12.  [3] tail -f /var/log/syslog&
13.  $ Mar  3 22:55:01 Firefly21 CRON[27163]: (root) CMD (if [ -x /etc/munin/plugins/apt_all
14.  Mar  3 22:55:14 Firefly21 dhclient: DHCPREQUEST of 192.168.1.18 on eth0 to 192.168.1.1
15.  Mar  3 22:55:14 Firefly21 dhclient: DHCPACK of 192.168.1.18 from 192.168.1.1
16.  Mar  3 22:55:14 Firefly21 dhclient: bound to 192.168.1.18 -- renewal in 1711 seconds.
```

# jobs

```
01.     $ jobs
02.     [3] +  Running                 tail -f /var/log/syslog
03.     [2] - Stopped                  vim snorkel.cpp
04.     [1]   Stopped                  vim test_servs.ksh
```

- Je kunt de terminal wel sluiten, proces blijft wel draaien.
- Handig bijvoorbeeld als je PhpStorm of sqldeveloper start op commandline om de stdout te kunnen lezen.

# Icecat

```
01.     touch log.txt
02.     tail -f log.txt &
03.     php icecat_import.php >> log.txt 2>&1
04.     ^C
```

# RTFM

- `$ ps --help (soms -h)`

# RTFM

- `$ ps --help (soms -h)`
- `man ps`

# RTFM

- `$ ps --help (soms -h)`
- `man ps`
- Als het anders is dan zegt zo'n programma dat meestal zelf..

# RTFM

- `$ ps --help (soms -h)`
- `man ps`
- Als het anders is dan zegt zo'n programma dat meestal zelf..

```
$ git bingo
git: 'bingo' is not a git command. See 'git --help'.
```

# RTFM

- `$ ps --help (soms -h)`
- `man ps`
- Als het anders is dan zegt zo'n programma dat meestal zelf..

```
$ git bingo
git: 'bingo' is not a git command. See 'git --help'.
```

- Geen vragen 'welke parameter was ... ook alweer?' nodig! :)

```
$ curl --help|grep user
     --ftp-alternative-to-user COMMAND  String to replace "USER [name]" (F)
 -n, --netrc         Must read .netrc for user name and password
 -U, --proxy-user USER[:PASSWORD]  Proxy user and password   <<<<<<<<<<<<<<<<<
 -u, --user USER[:PASSWORD]  Server user and password
     --tlsuser USER  TLS username
 -A, --user-agent STRING  User-Agent to send to server (H)
```

# Handige commands

```
01.     $ grep -i
02.     $ curl --help|grep -i post
03.      -d, --data DATA     HTTP POST data (H)
04.          --data-ascii DATA  HTTP POST ASCII data (H)
05.          --data-binary DATA  HTTP POST binary data (H)
06.          --data-urlencode DATA  HTTP POST data url encoded (H)
07.     $ curl --help| vim -
08.     $ awk
```

# Handige commands

```
$ grep -i
$ curl --help|grep -i post
 -d, --data DATA     HTTP POST data (H)
     --data-ascii DATA  HTTP POST ASCII data (H)
     --data-binary DATA  HTTP POST binary data (H)
     --data-urlencode DATA  HTTP POST data url encoded (H)
$ curl --help| vim -
$ awk
$ ps axu|grep www-data
www-data   893  0.0  1.0 356420 11164 ?        S    Mar05   0:00 /usr/sbin/apache2 -k s
www-data  6970  0.0  1.0 355816 10200 ?        S    Mar05   0:00 /usr/sbin/apache2 -k s
www-data 15792  0.0  1.0 356356 10184 ?        S    00:15   0:00 /usr/sbin/apache2 -k s
www-data 15800  0.0  0.9 356156  9280 ?        S    00:15   0:00 /usr/sbin/apache2 -k s
```

# Handige commands

```
01.     $ grep -i
02.     $ curl --help|grep -i post
03.      -d, --data DATA     HTTP POST data (H)
04.          --data-ascii DATA  HTTP POST ASCII data (H)
05.          --data-binary DATA  HTTP POST binary data (H)
06.          --data-urlencode DATA  HTTP POST data url encoded (H)
07.     $ curl --help| vim -
08.     $ awk
09.     $ ps axu|grep www-data |awk '{print $2}'
10.     1639
11.     1641
12.     1642
13.     1643
```

# Handige commands

```
$ grep -i
$ curl --help|grep -i post
 -d, --data DATA     HTTP POST data (H)
     --data-ascii DATA  HTTP POST ASCII data (H)
     --data-binary DATA  HTTP POST binary data (H)
     --data-urlencode DATA  HTTP POST data url encoded (H)
$ curl --help| vim -
$ awk
$ ps axu|grep www-data |awk '{print $2}'| kill -9
Usage: kill [-lL] [-n signum] [-s signame] job ...
   Or: kill [ options ] -l [arg ...]
```

# Handige commands

```
$ grep -i
$ curl --help|grep -i post
 -d, --data DATA     HTTP POST data (H)
     --data-ascii DATA  HTTP POST ASCII data (H)
     --data-binary DATA  HTTP POST binary data (H)
     --data-urlencode DATA  HTTP POST data url encoded (H)
$ curl --help| vim -
$ awk
$ ps axu|grep www-data |awk '{print $2}'| xargs kill -9
(staat gelijk aan kill -9 1639 1641 1642 1643)
```

# Handige commands

```
$ grep -i
$ curl --help|grep -i post
 -d, --data DATA     HTTP POST data (H)
     --data-ascii DATA  HTTP POST ASCII data (H)
     --data-binary DATA  HTTP POST binary data (H)
     --data-urlencode DATA  HTTP POST data url encoded (H)
$ curl --help| vim -
$ awk
$ ps axu|grep www-data |awk '{print $2}'| xargs echo kill -9
kill -9 1639 1641 1642 1643
```

# Handige commands

```
01.     $ grep -i
02.     $ curl --help|grep -i post
03.      -d, --data DATA     HTTP POST data (H)
04.          --data-ascii DATA  HTTP POST ASCII data (H)
05.          --data-binary DATA  HTTP POST binary data (H)
06.          --data-urlencode DATA  HTTP POST data url encoded (H)
07.     $ curl --help| vim -
08.     $ awk
09.     $ ps axu|grep www-data |awk '{print $2}'| xargs -n 1 echo kill -9
10.     kill -9 1639
11.     kill -9 1641
12.     kill -9 1642
13.     kill -9 1643
```

# Caveats

Wat doet dit:

```
$ echo pakje boter en melk | xargs -n 1 echo
```

# Caveats

Wat doet dit:

```
01.     $ echo pakje boter en melk | xargs -n 1 echo
02.     pakje
03.     boter
04.     en
05.     melk
```

# Caveats

Wat doet dit:

```
$ echo pakje boter en melk | xargs -n 1 echo
```

En deze..

```
$ echo "pakje boter en melk" | xargs -n 1 echo
```

# Caveats

Wat doet dit:

```
$ echo pakje boter en melk | xargs -n 1 echo
```

En deze..

```
01.  $ echo "pakje boter en melk" | xargs -n 1 echo
02.  pakje
03.  boter
04.  en
05.  melk
```

# Caveats

Wat doet dit:

```
$ echo pakje boter en melk | xargs -n 1 echo
```

En deze..

```
$ echo "pakje boter en melk" | xargs -n 1 echo
```

Gebruik:

```
01.  $ echo pakje boter en melk | xargs -0 -n 1 echo
02.  pakje boter en melk
```

- Spaties kunnen probleem zijn met windows files.

# Caveats

```
$ find ./ -name '*.php'| xargs -0 -n 1 php -l
```

# Caveats

```
$ find ./ -name '*.php'| xargs -0 -n 1 php -l
xargs: argument line too long
```

# Caveats

```
$ find ./ -name '*.php'| xargs -0 -n 1 php -l
xargs: argument line too long
```

Blijkbaar stuurt find geen NULL (\x00) na elke regel/newline. `printf "hello\nworld\n"` doet dat bijvoorbeeld wel.

# Caveats

```
01.     $ find ./ -name '*.php'| xargs -0 -n 1 php -l
02.     xargs: argument line too long
```

Daar is een optie voor:

# Caveats

```
$ find ./ -name '*.php'| xargs -0 -n 1 php -l
xargs: argument line too long
```

Daar is een optie voor:

```
$ find ./ -name '*.php' -print0 | xargs -0 -n 1 php -l
No syntax errors detected in ./library/Zend/Auth.php
No syntax errors detected in ./library/Zend/Text/Figlet/Exception.php
No syntax errors detected in ./library/Zend/Text/Table/Decorator/Unicode.php
No syntax errors detected in ./library/Zend/Text/Table/Decorator/Ascii.php
No syntax errors detected in ./library/Zend/Text/Table/Decorator/Interface.php
No syntax errors detected in ./library/Zend/Text/Table/Row.php
...
```

# Caveats

```
$ find ./ -name '*.php'| xargs -0 -n 1 php -l
xargs: argument line too long
```

Daar is een optie voor:

```
$ find ./ -name '*.php' -print0 | xargs -0 -n 1 php -l
No syntax errors detected in ./library/Zend/Auth.php
No syntax errors detected in ./library/Zend/Text/Figlet/Exception.php
No syntax errors detected in ./library/Zend/Text/Table/Decorator/Unicode.php
No syntax errors detected in ./library/Zend/Text/Table/Decorator/Ascii.php
No syntax errors detected in ./library/Zend/Text/Table/Decorator/Interface.php
No syntax errors detected in ./library/Zend/Text/Table/Row.php
...
```

Ook handige parameters van find:

```
-type d
-type f
```

Zie verder --help

# find en xargs

doe_leuk.php:

```
01.     <?php
02.     if ($_SERVER['argc'] !== 2) {
03.         print "not ok.. \n";
04.         print_r($_SERVER['argv']);
05.         // vind je het niet ok, kun je het laten weten met een status code != 0
06.         exit(127);
07.     }
08.
09.     $file = $_SERVER['argv'][1]; // zonder te checken
10.     $contents = file_get_contents($file);
11.
12.     // doe iets leuks
13.     print md5($contents);
14.     ?>
```

# find en xargs

```
$ find ./ -name '*.php' -print0 | xargs -0 -n 1 php doe_leuk.php
28cbe9b49d86d89fe37c5d5843f6c128
237eb58e784b4de99c3fd1bf53d1a90c
ef0dc94160055a3dc04ba2810a2667d0
46a46cbbceeb669d75d73af6bb4295f5
8ffccca664fb168ba725bacd02799243
...
```

# find en xargs

```
$ find ./ -name '*.php' -print0 | xargs -0 -n 1 php doe_leuk.php
28cbe9b49d86d89fe37c5d5843f6c128
237eb58e784b4de99c3fd1bf53d1a90c
ef0dc94160055a3dc04ba2810a2667d0
46a46cbbceeb669d75d73af6bb4295f5
8ffccca664fb168ba725bacd02799243
...
$ find ./ -name '*.php' -print0| xargs -0 -n 1 php doe_leuk.php extra_param
not ok..
Array
(
    [0] => doe_leuk.php
    [1] => extra_param
    [2] => ./library/Zend/Auth.php
)
```

# find en xargs

```
01.     $ find ./ -name '*.php' -print0 | xargs -0 -n 1 php doe_leuk.php
02.     28cbe9b49d86d89fe37c5d5843f6c128
03.     237eb58e784b4de99c3fd1bf53d1a90c
04.     ef0dc94160055a3dc04ba2810a2667d0
05.     46a46cbbceeb669d75d73af6bb4295f5
06.     8ffccca664fb168ba725bacd02799243
07.     ...
        $ find ./ -name '*.php' -print0| xargs -0 -n 1 php doe_leuk.php extra_param
```

- xargs stopt gelijk op basis van de return code.

# find en xargs

```
01.     $ find ./ -name '*.php' -print0 | xargs -0 -n 1 php doe_leuk.php
02.     28cbe9b49d86d89fe37c5d5843f6c128
03.     237eb58e784b4de99c3fd1bf53d1a90c
04.     ef0dc94160055a3dc04ba2810a2667d0
05.     46a46cbbceeb669d75d73af6bb4295f5
06.     8ffccca664fb168ba725bacd02799243
07.     ...
        $ find ./ -name '*.php' -print0| xargs -0 -n 1 php doe_leuk.php extra_param
```

- xargs stopt gelijk op basis van de return code.
- php -l doet dat ook, zodra een PHP file syntax NOK is.

# find en xargs

```
$ find ./ -name '*.php' -print0 | xargs -0 -n 1 php doe_leuk.php
28cbe9b49d86d89fe37c5d5843f6c128
237eb58e784b4de99c3fd1bf53d1a90c
ef0dc94160055a3dc04ba2810a2667d0
46a46cbbceeb669d75d73af6bb4295f5
8ffccca664fb168ba725bacd02799243
...
$ find ./ -name '*.php' -print0| xargs -0 -n 1 php doe_leuk.php extra_param
```

- xargs stopt gelijk op basis van de return code.
- php -l doet dat ook, zodra een PHP file syntax NOK is.
- als je bijvoorbeeld de return code wilt negeren (kan vast simpeler)

```
find ./ -name '*.php' -print0|
xargs -0 -n 1 sh -c "php doe_leuk.php <file_hier> extra_param; return 0"
```

# find en xargs

```
$ find ./ -name '*.php' -print0 | xargs -0 -n 1 php doe_leuk.php
28cbe9b49d86d89fe37c5d5843f6c128
237eb58e784b4de99c3fd1bf53d1a90c
ef0dc94160055a3dc04ba2810a2667d0
46a46cbbceeb669d75d73af6bb4295f5
8ffccca664fb168ba725bacd02799243
...
$ find ./ -name '*.php' -print0| xargs -0 -n 1 php doe_leuk.php extra_param
```

- xargs stopt gelijk op basis van de return code.
- php -l doet dat ook, zodra een PHP file syntax NOK is.
- als je bijvoorbeeld de return code wilt negeren (kan vast simpeler)

```
find ./ -name '*.php' -print0|
xargs -I{} -0 -n 1 sh -c "php doe_leuk.php <file_hier> extra_param; return 0"
```

# find en xargs

```
$ find ./ -name '*.php' -print0 | xargs -0 -n 1 php doe_leuk.php
28cbe9b49d86d89fe37c5d5843f6c128
237eb58e784b4de99c3fd1bf53d1a90c
ef0dc94160055a3dc04ba2810a2667d0
46a46cbbceeb669d75d73af6bb4295f5
8ffccca664fb168ba725bacd02799243
...
$ find ./ -name '*.php' -print0| xargs -0 -n 1 php doe_leuk.php extra_param
```

- xargs stopt gelijk op basis van de return code.
- php -l doet dat ook, zodra een PHP file syntax NOK is.
- als je bijvoorbeeld de return code wilt negeren (kan vast simpeler)

```
find ./ -name '*.php' -print0|
xargs -I{} -0 -n 1 sh -c "php doe_leuk.php {} extra_param; return 0"
```

# find en xargs

```
$ find ./ -name '*.php' -print0 | xargs -0 -n 1 php doe_leuk.php
28cbe9b49d86d89fe37c5d5843f6c128
237eb58e784b4de99c3fd1bf53d1a90c
ef0dc94160055a3dc04ba2810a2667d0
46a46cbbceeb669d75d73af6bb4295f5
8ffccca664fb168ba725bacd02799243
...
$ find ./ -name '*.php' -print0| xargs -0 -n 1 php doe_leuk.php extra_param
```

- xargs stopt gelijk op basis van de return code.
- php -l doet dat ook, zodra een PHP file syntax NOK is.
- als je bijvoorbeeld de return code wilt negeren (kan vast simpeler)

```
find ./ -name '*.php' -print0|
xargs -I{} -0 -n 1 sh -c "echo md5 van {}; php doe_leuk.php {}; return 0"
```

# find en xargs

```
sh -c "<string>" is wel geinig.
```

man sh geeft:

```
-c    Read commands from the command_string operand instead of from the standard input.
```

# (e)grep

```
$ ps axufw|egrep "apache2|mysql" | grep -v grep
```

# (e)grep

```
01.  $ ps axufw|egrep "apache2|mysql" | grep -v grep
02.  mysql     1524  0.0  0.1 484352 43804 ?        Ssl  mrt03   0:22 /usr/sbin/mysqld
03.  root      1636  0.0  0.0 343192 18524 ?        Ss   mrt03   0:01 /usr/sbin/apache2 -k s
04.  www-data  1639  0.0  0.0 151940  3396 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
05.  www-data  1640  0.0  0.0 343328  8220 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
06.  www-data  1641  0.0  0.0 343328  8220 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
07.  www-data  1642  0.0  0.0 343328  8220 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
08.  www-data  1643  0.0  0.0 343328  8220 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
09.  www-data  1644  0.0  0.0 343328  8220 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
10.  www-data 31196  0.0  0.0 343328  8224 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
```

# (e)grep

```
01.     $ ps axufw|egrep "apache2|mysql" | grep -v grep
02.     mysql     1524  0.0  0.1 484352 43804 ?        Ssl  mrt03   0:22 /usr/sbin/mysqld
03.     root      1636  0.0  0.0 343192 18524 ?        Ss   mrt03   0:01 /usr/sbin/apache2 -k s
04.     www-data  1639  0.0  0.0 151940  3396 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
05.     www-data  1640  0.0  0.0 343328  8220 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
06.     www-data  1641  0.0  0.0 343328  8220 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
07.     www-data  1642  0.0  0.0 343328  8220 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
08.     www-data  1643  0.0  0.0 343328  8220 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
09.     www-data  1644  0.0  0.0 343328  8220 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
10.     www-data 31196  0.0  0.0 343328  8224 ?        S    mrt03   0:00  \_ /usr/sbin/apache2
```

- egrep schijnt hetzelfde te zijn als `grep -E`
- grep -v is omgedraaide grep.

# sed

```
01.     $ w|sed s/USER/HERO/g
02.      01:35:28 up 15:04,  2 users,  load average: 0,15, 1,80, 1,91
03.     HERO      TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
04.     trigen   tty7     :0               21:22   15:04m  5:28   0.10s init --user
05.     trigen   pts/0    :0               01:10    0.00s  0.01s  0.00s w
```

# sed

```
$ w|sed s/USER/HERO/g
 01:35:28 up 15:04,  2 users,  load average: 0,15, 1,80, 1,91
HERO     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
trigen   tty7     :0               21:22   15:04m  5:28   0.10s init --user
trigen   pts/0    :0               01:10    0.00s  0.01s  0.00s w
```

- sed kan echt heel veel, leuke link naar oneliners.

# sed

```
01.  $ w|sed s/USER/HERO/g
02.   01:35:28 up 15:04,  2 users,  load average: 0,15, 1,80, 1,91
03.  HERO     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
04.  trigen   tty7     :0               21:22   15:04m  5:28   0.10s init --user
05.  trigen   pts/0    :0               01:10    0.00s  0.01s  0.00s w
```

- sed kan echt heel veel, leuke link naar oneliners.
- sed '/trigen/d' (delete line ONLY for lines which contain "trigen")

# sed

```
$ sed '/tty/s/trigen/superbeer/g'
 01:35:28 up 15:04,  2 users,  load average: 0,15, 1,80, 1,91
HERO     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
superbeer    tty7     :0               21:22   15:04m  5:28   0.10s init --user
trigen   pts/0    :0               01:10    0.00s  0.01s  0.00s w
```

- sed kan echt heel veel, leuke link naar oneliners.
- sed '/trigen/d' (delete line ONLY for lines which contain "trigen")
- sed '/tty/s/trigen/superbeer/g' (substitute "trigen" with "superbeer" ONLY for lines which contain "tty")

# sed

```
$ sed '/tty/s!/trigen/superbeer/g'
 01:35:28 up 15:04,  2 users,  load average: 0,15, 1,80, 1,91
HERO     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
trigen   tty7     :0                21:22   15:04m  5:28   0.10s init --user
superbeer pts/0    :0                01:10    0.00s  0.01s  0.00s w
```

- sed kan echt heel veel, leuke link naar oneliners.
- sed '/trigen/d' (delete line ONLY for lines which contain "trigen")
- sed '/tty/s/trigen/superbeer/g' (substitute "trigen" with "superbeer" ONLY for lines which contain "tty")
- sed '/tty/!s/trigen/superbeer/g' (substitute "trigen" with "superbeer" EXCEPT for lines which contain "tty")

# vim

- find ./ -name '*.php' | less
- control+u, control+d (up/down, zelfde als in vim)
- find ./ -name '*.php' | vim -
- In shell: `set -o vi`, command mode

# bash

- ctrl+r (reverse search), je hebt geloof ik ook de andere kant op
- sidenode: ctrl+s freezed sommige shells, ctrl+q voor unfreeze!
- verder emacs: ctrl+a, ctrl+e
- soort van command mode (eenmalig 'escape', w, e.d.)
- command escapen met # (in command mode)

# Processes in linux

- Elk proces heeft een PID.
- In je shell kun je processen ook benaderen als jobs. (eerder gezien)
- Je kunt bekijken wat een proces low level uitvoert met `strace`

# strace voorbeeld

- strace bij aanroepen proces

```
01.  $ strace w 2>&1 | head -n 10
02.  execve("/usr/bin/w", ["w"], [/* 33 vars */]) = 0
03.  brk(0)                                      = 0xb3b000
04.  access("/etc/ld.so.nohwcap", F_OK)          = -1 ENOENT (No such file or directory)
05.  mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc0a33a1000
06.  access("/etc/ld.so.preload", R_OK)          = -1 ENOENT (No such file or directory)
07.  open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
08.  fstat(3, {st_mode=S_IFREG|0644, st_size=45959, ...}) = 0
09.  mmap(NULL, 45959, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc0a3395000
10.  close(3)                                    = 0
11.  access("/etc/ld.so.nohwcap", F_OK)          = -1 ENOENT (No such file or directory)
```

# strace voorbeeld

- strace bij aanroepen proces
- strace op bestaand proces (volgende slide)

```
$ ps axufw
root      1636  0.0  0.0 343192 18524 ?        Ss   10:30   0:00 /usr/sbin/apache2 -k start
www-data  1639  0.0  0.0 151940  3396 ?        S    10:30   0:00  \_ /usr/sbin/apache2 -k sta
www-data  1640  0.0  0.0 343328  8220 ?        S    10:30   0:00  \_ /usr/sbin/apache2 -k sta
www-data  1641  0.0  0.0 343328  8220 ?        S    10:30   0:00  \_ /usr/sbin/apache2 -k s
www-data  1642  0.0  0.0 343328  8220 ?        S    10:30   0:00  \_ /usr/sbin/apache2 -k sta
www-data  1643  0.0  0.0 343328  8220 ?        S    10:30   0:00  \_ /usr/sbin/apache2 -k sta
www-data  1644  0.0  0.0 343328  8220 ?        S    10:30   0:00  \_ /usr/sbin/apache2 -k sta
www-data 31196  0.0  0.0 343328  8224 ?        S    19:07   0:00  \_ /usr/sbin/apache2 -k sta
```

# strace voorbeeld

- strace bij aanroepen proces
- strace op bestaand proces (volgende slide)

```
01.     $ ps axufw
02.     ...
03.     $ sudo strace -p 1641
04.     Process 1641 attached
05.     accept4(3,
```

# strace voorbeeld

- strace bij aanroepen proces
- strace op bestaand proces (volgende slide)

```
01.     $ ps axufw
02.     ...
03.     $ sudo strace -p 1641
04.     Process 1641 attached
05.     accept4(3,
```

- Die doet in mijn geval niet zoveel niet zoveel. (Weinig traffic op mijn locale dev bak.)

# strace

- strace -o logfile.txt
- strace -s
- strace -fF
- strace --help

# strace demo

- Boven in een PHP file

```
01.     file_put_contents('/tmp/hoi.txt', getmypid());
02.     sleep(10);
```

- strace -fF -p `cat /tmp/hoi.txt` -o output.log, of:
- strace -fF -p $(cat /tmp/hoi.txt) -o output.log

# strace

- Prefereer de -o output (of in combinatie met -ff) t.o.v.:

```
strace -fF -p $(cat /tmp/hoi.txt) > output.log 2>&1
```

# Nog wat commands

- ls -althr

# Nog wat commands

- ls -althr
- ls -althrS

# Nog wat commands

- ls -althr
- ls -althrS
- du -h --max-depth=1

# Nog wat commands

- ls -althr
- ls -althrS
- du -h --max-depth=1
- df -h

# Nog wat commands

- ls -althr
- ls -althrS
- du -h --max-depth=1
- df -h
- free -m

# Nog wat commands

- ls -althr
- ls -althrS
- du -h --max-depth=1
- df -h
- free -m
- screen

# Nog wat commands

- ls -althr
- ls -althrS
- du -h --max-depth=1
- df -h
- free -m
- screen
- tail -f

# Nog wat commands

- ls -althr
- ls -althrS
- du -h --max-depth=1
- df -h
- free -m
- screen
- tail -f
- tail -n 10

# Nog wat commands

- head -n 10

# Nog wat commands

- head -n 10
- wc -l (line count)

# Nog wat commands

- head -n 10
- wc -l (line count)
- wc -c (bytes)

# Nog wat commands

- head -n 10
- wc -l (line count)
- wc -c (bytes)
- curl

# Nog wat commands

- head -n 10
- wc -l (line count)
- wc -c (bytes)
- curl
- sudo -u <user> [ -g <group> ] *iets*

# Nog wat commands

- head -n 10
- wc -l (line count)
- wc -c (bytes)
- curl
- sudo -u <user> [ -g <group> ] *iets*
- top / htop

# Nog wat commands

- head -n 10
- wc -l (line count)
- wc -c (bytes)
- curl
- sudo -u <user> [ -g <group> ] *iets*
- top / htop
- telnet

# Nog wat commands

- head -n 10
- wc -l (line count)
- wc -c (bytes)
- curl
- sudo -u <user> [ -g <group> ] *iets*
- top / htop
- telnet
- lsof -i -P -n

# Nog wat commands

- head -n 10
- wc -l (line count)
- wc -c (bytes)
- curl
- sudo -u <user> [ -g <group> ] *iets*
- top / htop
- telnet
- lsof -i -P -n

# Dependencies in linux

- `file <iets>` zegt wat het is, een script, symlink, character special
- `ls -al` laat soms ook al e.e.a. zien.
- `whereis grep`
- `ldd <iets>`

# En nog veel meer..

permissies, chmod, chown, sudo, sudoers, file attributes screen signals

hoe files werken in linux