

Linux & Bash

Michiel Noback

12 July 2016

Contents of this presentation

- ▶ man pages (arrgghhh)
- ▶ file system operations (continued)
- ▶ finding & filtering stuff (egrep, find, sed)
- ▶ piping
- ▶ redirecting

Part 1: Man pages

man

- ▶ Every commandline tool on the Linux (and Mac) system has a `man` page. It describes its purpose, flags, arguments and usage scenarios.
- ▶ However, they are often quite unreadable. . . Googling may prove faster than reading the man page
- ▶ But it's there and you should be aware of it
- ▶ Use `man <command>` to read it and `/<keyword>` to look for a certain topic

NAME

ls -- list directory contents

SYNOPSIS

ls [-ABCFGHLOPRSTUW@abcdefghijklmnopqrstuvwxyz1] [file ...]

DESCRIPTION

For each operand that names a file of a type other than directory, ls displays the file's name. If an operand names a file of type directory, ls displays the names of the files in the directory.

If no operands are given, the contents of the current directory are listed. If the operand `./` is given, the contents of the current directory are listed. If the operand `..` is given, the contents of the parent directory are listed. Directory and non-directory operands are sorted in the order specified by the `-A` option.

The following options are available:

-@ Display extended attribute keys and sizes in the output.

manpage navigation

- ▶ space = next page
- ▶ enter = next line
- ▶ q = quit
- ▶ / = find
- ▶ arrow keys = line up/down

Part 2: File system operations

Creating

- ▶ create a file: `touch <file name>`
- ▶ create a directory: `mkdir <dir name>`
- ▶ create a folder and subfolders in one go, e.g. `mkdir -p /foo/bar/baz`

Copying

- ▶ copy file from one folder to another:
- ▶ `cp <source> <dest folder or name>`

```
cp some_text ~/Desktop/tmp #copy with same name  
cp some_text ~/Desktop/tmp/some_text2.txt #copy with new name
```

Renaming and moving

- ▶ use `mv` to rename a file/folder or move it to another location
- ▶ It has two forms
 - ▶ `mv <source> <target>` is a rename
 - ▶ `mv <source> <target folder>` moves to another location
- ▶ The difference with `cp` is that `mv` will remove the source file/folder!

Deleting (without undo button!)

- ▶ use `rmdir` to delete empty folders
- ▶ use `rm` to delete files and folders; take extra care with flags
 - ▶ `-R` delete recursive - also subfolders
 - ▶ `-f` delete without prompting for confirmation
- ▶ **!! especially `rm -Rf` is very dangerous; use it with extreme caution because there is no undo!!**

Part 3: Redirecting and combining

A tale of 3 streams

The terminal has three streams:

- ▶ `stdin` (0) or the input stream (usually what you type on the keyboard, but it can also be the output from some other program)
- ▶ `stdout` (1) or the standard output stream (terminal output)
- ▶ `stderr` (2) or the error output stream

The numbers can be used when redirecting.

A simple redirect

- ▶ Simply use `> log.txt` when you to redirect all output from a program to a file (file will be overwritten every time the command is run)
- ▶ Use `>> log.txt` when you want to append to the file instead of overwriting it

Give it a try; type `echo Hello World > hello.log` in the terminal

stdout to stdin

- ▶ If you want to push the output stream of one program to the input of another, use piping: |
- ▶ This example pipes the result from echo to sed:

```
echo "Hello World" | sed s/Hello/Bye/
```

```
## Bye World
```

This seems rather trivial but

!!piping is one of the major strengts of the terminal!!

The error stream

- ▶ Programs often have their normal output, but occasionally also an error message that you may not want to have in the same place
- ▶ In those cases, you could use `2>` to redirect the error messages
- ▶ The following slide demonstrate these kind of scenarios

streams_demo.py

This Python script (streams_demo.py) demonstrates the use of all three streams

```
#!/usr/local/bin/python3
import sys

# default printing to stdout
print("My input was \"", sys.stdin.readline().strip(), '"')

# error message to stderr
print("Oh oh, we have a problem, Houston", file = sys.stderr)

# be specific about stdout
print("Ahh, solved it. Bye!", file=sys.stdout)
```

Streams demo: No redirect

Basic usage without redirect

```
echo "Hello Stream World" | ./data/streams_demo.py
```

```
Oh oh, we have a problem, Houston  
My input was "Hello Stream World"  
Ahh, solved it. Bye!
```

Notice that the output from the different streams is not in the correct order; this is an RMarkdown artifact!

Streams demo: Redirect stderr

```
echo "Hello Stream World" | ./data/streams_demo.py 2> ./dat
```

My input was "Hello Stream World"
Ahh, solved it. Bye!

and the error log contains this

```
cat ./data/error.log
```

Oh oh, we have a problem, Houston

Streams demo: Redirect stdout

```
echo "Hello Stream World" | ./data/streams_demo.py 1> ./dat
```

Oh oh, we have a problem, Houston

and the normal log contains this

```
cat ./data/normal.log
```

My input was "Hello Stream World"

Ahh, solved it. Bye!

Streams demo: Redirect both streams

```
echo "Hello Stream World" | ./data/streams_demo.py \  
1> ./data/normal.log 2> ./data/error.log
```

and the normal log contains this

```
cat ./data/normal.log
```

```
My input was "Hello Stream World"  
Ahh, solved it. Bye!
```

and error this

```
cat ./data/error.log
```

```
Oh oh, we have a problem, Houston
```

If you just want to get rid of stderr

- ▶ If you know there may be some error but are not interested in them, redirect to `/dev/null`
- ▶ This is the Linux black hole of bytes.

Part 4: Finding & filtering stuff

egrep

- ▶ egrep, or extended grep, is used to Get Regular ExPressions in text file(s) or streams
- ▶ The regular expression syntax is (almost) the same as in Python
- ▶ Some examples, given the previously used text file `some_text`:

Linux is a Unix-like and mostly POSIX-compliant[12] computer operating system (OS) assembled under the model of free and open-source software development and distribution.

The defining component of Linux is the Linux kernel,[13] an operating system kernel first released on October 5, 1991 by Linus Torvalds.[14][15]

The Free Software Foundation uses the name GNU/Linux to describe the operating system, which has led to some controversy.[16][17]

This `egrep` command looks for the word “develop” in each line in the file `some_text` and prints the line contents:

```
egrep 'develop' ./data/some_text
```

open-source software development and distribution.

Linux was originally developed as a free operating system

Here, two consecutive citations are searched (e.g) in a line, and output also includes the line number:

```
egrep -n '(\[d{2}\]){2}' ./data/some_text
```

6:Linux Torvalds.[14][15]

9:controversy.[16][17]

20:virtually all fastest supercomputers,[20][21] but is u

21:only around 1.6% of desktop computers[22][23] when not

29:facility automation controls, televisions,[26][27] vic

egrep options

egrep has *many* options; read the manual is a good idea here. Here are the most commonly used options:

- ▶ `-c` counts the occurrences instead of printing them
- ▶ `-n` give line numbers of matched lines
- ▶ `-R` recursively search in subfolders (given a folder)
- ▶ `--color` highlights the found substring on each line
- ▶ `-l` only lists the filenames of files where matches were found

find

- ▶ Find is a very powerful tool to... find stuff.
- ▶ an example best explains

```
find ~/Dropbox -maxdepth 3 -name '*java*' -type f | head -n
```

```
## /Users/michiel/Dropbox/courses/BFV_3_Thema10_WebBased1/V
## /Users/michiel/Dropbox/courses/BFV_3_Thema10_WebBased1/V
## /Users/michiel/Dropbox/courses/BFV_3_Thema10_WebBased1/V
## /Users/michiel/Dropbox/eBooks/tech/javafx-swing.pdf
```

Here, objects of type file (`-type f`) are searched within the Dropbox folder recursively, to a maximum depth of 3 folders (`-maxdepth 3`) where the name holds the substring 'java' (`-name '*java*'`). For convenience, I kept only the first 4 objects

sed (stream editor)

- ▶ sed can be used to modify text streams
- ▶ we'll only deal with its simplest form: substitution

```
echo "foo bar baz" | sed 's/ /+/'  
echo "foo bar baz" | sed 's/ /+/g'
```

```
## foo+bar baz  
## foo+bar+baz
```

Note the use of the `g` flag to make the matching process global

There is so much more sed

- ▶ Use the -E switch to get extended regular expressions
- ▶ Use & to re-use the found pattern
- ▶ Use grouping parentheses and \1, \2 to get parts of a pattern for re-use
- ▶ Have a look at, for instance, <http://www.grymoire.com/Unix/Sed.html> to check out more on advanced finding and replacing.

```
echo Sunday | sed 's/[una]/(&&)/g'
echo "Postal codes look like: 9999AA or 8888 BB" \
| sed -E 's/([0-9]{4}) ?([A-Za-z]{2})/(AREA=\1)(STREET=\2)/'
```

```
## S(uu)(nn)d(aa)y
```

```
## Postal codes look like: (AREA=9999)(STREET=AA) or (AREA=
```

Putting it all together

- ▶ Use of find with -exec, pipe and sed
- ▶ This quite an extensive example. Can you figure out what's going on?

```
find ~/Dropbox -maxdepth 2 -name '*java*' -type f -exec ls  
| sed 's|/Users/michiel/Dropbox|.|'
```

```
## 240 ./Java/hamcrest-core-1.3-javadoc.jar  
## 916 ./Java/junit-4.12-javadoc.jar  
## 104 ./Java/system-rules-1.8.0-javadoc.jar
```