UNIX tools

Table of contents

Please!

Do NOT take literal notes. I will be handing a printed copy of everything

DO write down questions about the use of the tools, we'll solve them at the end

I'll be passing slides very fast, so stop me if you have any question

I know that I am going to give you too much information. Don't worry if there is something strange that you don't understand.

Our input file

```
one     1
two     2
                      <- blank line
THREE   3
four    4
three   3
eleven 11
twelve 12
```

‘grep’ for lines

‘grep’ will output the lines which match to your request

This tool is very very simple and limited, but with some thinking you can use it for almost any pattern matching

‘grep’ for lines

```
$ grep “one” file.txt
one    1


$ grep “ee” file.txt
three 3
```

usage: grep “pattern” file

inverted ‘grep’

```
$ grep -v “three” file.txt
one     1
two     2


THREE   3
four    4
eleven  11
twelve  12
```

ignore case

```
$ grep -i “three” file.txt
THREE 3
three 3
```

(notice that the output always appears in its original uppercase form)

grep patterns

```
$ grep "[12]" file.txt
one     1
two     2
eleven 11
twelve 12
```

"1 or 2"

grep patterns

```
$ grep "[2-4]" file.txt
two    2
THREE  3
four   4
three  3
twelve 12
```

"numbers from 2 to 4"

grep patterns

```
$ grep “t[hw]” file.txt

two    2

three  3

twelve 12
```

‘t’ followed by a ‘h’ or ‘w’

^begin and end$

```
$ grep “^o” file.txt
one    1


$ grep “1$” file.txt
one    1
eleven 11
```

color grep

```
$ grep --color=auto “3” file.txt
THREE 3
three 3
```

(Add this to your ~/.bashrc)

```
alias grep=”grep --color=auto”
```

remove empty lines

```
$ grep “.” file.txt
one 1
two 2
THREE 3
four 4
three 3
eleven 11
twelve 12
```

‘awk’ for columns

‘awk’ can be used in the same fashion as ‘grep’, but it really is a full featured programming language

It is very useful to parse column formatted files

$0 is the whole line, $1 is the first column, and so on

‘awk’ for columns

```
$ awk ‘{print $1}’ file.txt
one
two

THREE
four
three
eleven
twelve
```

reorder columns

```
$ awk '{print $2,$1}' file.txt
1 one
2 two

3 THREE
4 four
3 three
11 eleven
12 twelve
```

A new data file

```
name1 chr1 15000
name2 chr1 20000
name3 chr1 20000
name4 chr3 90800
name5 chr5 22000
name6 chr6 73000
```

kind of duplicates

```
$ awk 'seen[$2 $3]++ == 1'
file2.txt
```

```
name3 chr1 20000
```

The identifier is different but it refers to the same position as “name2 chr1 20000”, so it is a duplicate

numeric ranges

```
$ awk ‘$3 > 50000 {print $1}’
file2.txt

name4

name6
```

If the third column is over 50,000 we print the first column

‘sed’ for substituting

‘sed’ easily replaces patterns in a file

It is a very complete tool, however, I will only present the substitution part

Usage:

```
$ sed “s/old/new/g” file
```

replace with ‘sed’

```
$ sed "s/name/id_/g" file2.txt
id_1 chr1 15000
id_2 chr1 20000
id_3 chr1 20000
id_4 chr3 90800
id_5 chr5 22000
id_6 chr6 73000
```

‘sed’ only some lines

```
$ sed "1,2s/name/id_/g"
file2.txt

id_1 chr1 15000

id_2 chr1 20000

name3 chr1 20000

name4 chr3 90800

name5 chr5 22000

name6 chr6 73000
```

'sed' in place

```
$ sed -i "s/name/id_/g"
file2.txt
```

It will overwrite file2.txt with the new contents. No terminal output is generated.

Other tools

'sort' sorts files

'wc' is a char/word/line counter, 'seq' builds sequences

'uniq' removes adjacent duplicate lines

'head' and 'tail'

'cut' and 'paste' columns

sort

```
$ sort file.txt
THREE 3
eleven 11
four 4
one 1
three 3
twelve 12
two 2
```

sort

```
$ awk {'print $2,$1'} file.txt
| sort

1 one

11 eleven

12 twelve

2 two

3 THREE

3 three

4 four
```

sort -g

```
$ awk {'print $2,$1'} file.txt
| sort -g

1 one

2 two

3 THREE

3 three

4 four

11 eleven

12 twelve
```

wc

```
$ wc file.txt
7       14       55 file.txt
```

Useful to count the number of files in a directory (always subtract 1 for the "total" line)

```
$ ls -l | wc -l
14
```

head and tail

```
$ head -n 1 file.txt
one     1

$ tail -n 2 file.txt
eleven 11
twelve 12

$ tail -f output.txt
11:54 running s_a_19982
11:57 running s_a_19983
11:59 running s_a_19984
```

seq

```
$ seq -s ", " -w 0 3 20
00, 03, 06, 09, 12, 15, 18
```

‘seq’ generates numeric sequences, separated by a specific character, given a start, end and step

paste

```
$ paste file2.txt file2.txt

name1 chr1 15000 name1 chr1 15000

name2 chr1 20000 name2 chr1 20000

name3 chr1 20000 name3 chr1 20000

name4 chr3 90800 name4 chr3 90800

name5 chr5 22000 name5 chr5 22000

name6 chr6 73000 name6 chr6 73000
```

‘paste’ concatenates different files, line by line, in columns

cut

```
$ cut -b 12-13 file2.txt

15

20

20

90

22

73
```

'cut' is sometimes better than awk because it can work with single bytes, not columns (i.e. PDB-style)

Bash scripts

Rule 1: use pipes

```
$ ls -l | wc -l
```

Rule 2: use iterations

```
$ for i in data*; do echo $i;
done
```

Rule 3: if you can’t do it with 1 and 2, then call an external program

```
$ for i in data*; do perl -e
"print uc $i"; done
```

variable usage

To declare a variable

```
$ beer="Lowenbrau"; echo $beer
Lowenbrau
```

To evaluate it

```
$ echo 2+2
2+2
$ echo $((2+2))
4
```

run commands

```
$ result=`head -n 1 file.txt`
$ echo $result
one    1


$ echo $result | awk {'print
$1'}
one
```

quotation mark mayhem

```
$ beer=”Lowenbrau”


$ echo “I like $beer”
I like Lowenbrau
$ echo 'I like $beer'
I like $beer
$ echo "I like \"$beer\""
I like "Lowenbrau"
$ echo "I like `seq -s \",\" 10` beers"
I like 1,2,3,4,5,6,7,8,9,10 beers
```

conditionals

Only if the file exists

```
$ if [ -e “oldfile.txt” ];
then rm oldfile.txt; fi
```

Only if some condition

```
$ data=`head -n 1 file.txt | awk
{‘print $1’}`

$ if [ “$data” == “one” ]; then echo
“yes”; else echo “no”; fi
```

our data set

```
chr1/
chr2/
chr3/
...
chr22/
chrX/
chrY/
index.txt
```

let's iterate

I want to run some script on all the existing chromosomes

```
$ for i in chr*; do cd $i; run_software.sh; cd ..; done
```

Only on the numerical ones

```
$ for i in chr[0-9][0-9]; do cd $i; run_software.sh; cd ..; done
```

let's iterate

I want to create new folders (now the brackets don't work, as the files don't exist yet!)

We need to use brace expansion

```
$ for i in newdir{0..9}; do
mkdir $i; done

$ ls

newdir0 newdir1 newdir2
newdir3 ... ... newdir9
```

quick keystrokes

<Ctrl-a>	Goto beginning of line
<Ctrl-e>	Goto end of line
<Alt-f>	Move forward a word
<Alt-b>	Move back a word
<Ctrl-u>	Delete from BOL to here
<Ctrl-k>	Delete from here to EOL
<Ctrl-w>	Delete a word backwards
<Alt-d>	Delete a word forward
<Alt-t>	Swap current word with prev

just more tricks

Run a process in background

```
$ firefox &
[1] 7240
```

End it

```
$ kill %1
[1]+ Done firefox
```

Unattach a process from a terminal

```
$ nohup wget http://ubuntu.com/ubuntu.iso &
nohup: se ignora la entrada y se añade la salida a
«nohup.out»
```

redirections

You already know

```
$ cat input.txt > output.txt
```

Append to a file

```
$ cat input2.txt >> output.txt
```

To use the command line as if it were a file

```
$ cat "Hello"

cat: Hello: No such file or directory

$ cat <<< "Hello"

Hello
```

the final secret

Do you stil ssh the old way?

```
$ ssh username@mmb.pcb.ub.es -X
Enter password:
```

First, create ~/.ssh/config

```
Host mmb
    HostName mmb.pcb.ub.es
    User carlesfe
    ForwardX11 yes
```

the final secret

Here is how to avoid passwords

```
$ ssh-keygen -t dsa
```

(Use an empty password)

```
$ cat .ssh/id_dsa.pub | ssh mmb “cat >> .ssh/
authorized_keys2”
```

And now...

```
$ ssh mmb

carlesfe@mmb:~$ chmod 0600 .ssh/authorized_keys2
```

Repeat the ‘ssh-keygen’ command in mmb, append the ‘id_rsa.pub’ to mmb’s ‘authorized_keys2’ and you will be able to ssh to the nodes without a password. Really useful! BE CAREFUL: Don’t leave your session unattended, as no password is required to login!

vim

Remember always to use ‘vim’ instead of ‘vi’

Enable syntax highlighting

vim is huge, but these few hints will do

moving (command mode)

`:256`	Go to line 256
`%`	Go to the matching bracket
`O`	Start a line above
`o`	Start a line below
`>`	Indent line
`<`	Unindent line
`u`	Undo
`:redo`	Redo

Copy and paste (command)

yy Copy current line

dd Cut current line

p Paste

P Paste before or above

v Enter visual mode

y, d, p Copy, Cut, Paste block

Search and replace

`/test` Search for “test”

`n` Next match

`N` Previous match

`:1,$s/old/new/g` Replace in the whole file (like in ‘sed’)

`gd` Go to the variable definition

`:%!command` run ‘command’ and paste the output inside vim

Useful config

```
set tabstop=4
set shiftwidth=4
set ai                             // autoindent
set si                             // smart indent
syntax enable                      // color
set number                         // line numbers
set backspace=indent,eol,start     // smart backspace
set ignorecase                     // for searches
set showmatch                      // highlight searches
set nocompatible
set noexpandtab
```

Use ":set paste" before pasting text to avoid getting too many spaces, then later ":set nopaste"

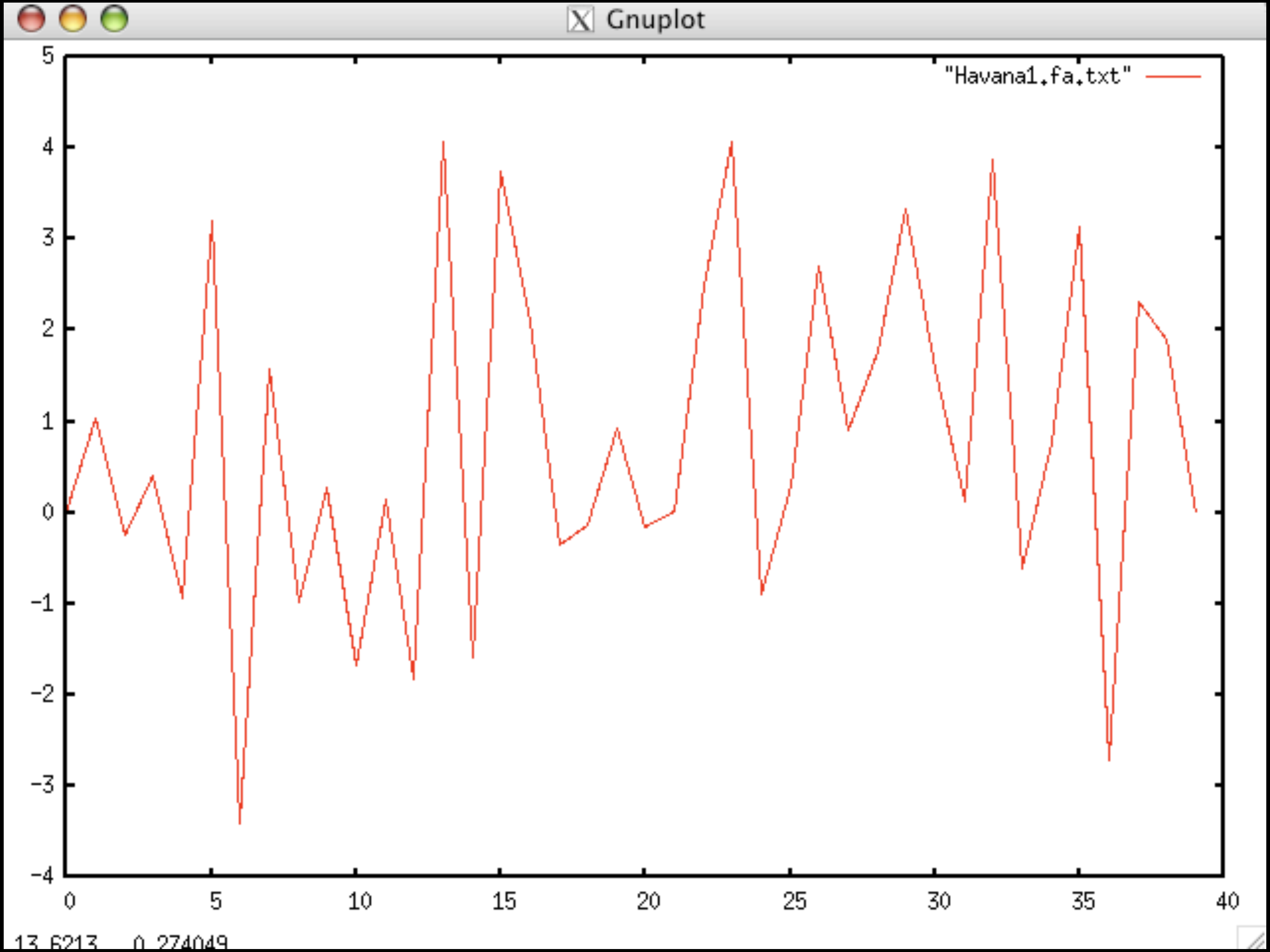
gnuplot

VERY easy to use but confusing to master

```
$ gnuplot
```

(Blah, blah)

```
gnuplot> plot “data.txt” w lines
```


Gnuplot
"Havana1.fa.txt"
5
4
3
2
1
0
-1
-2
-3
-4
0
5
10
15
20
25
30
35
40
13.6213   0.274049

long line

```
gnuplot> plot “1.txt” w lines lw 2
title “First”, “2.txt” w imp title
“Second”, “2columns.txt” using 1:2 w
points ps 4 notitle;
```

w with (plot type)

lw line width

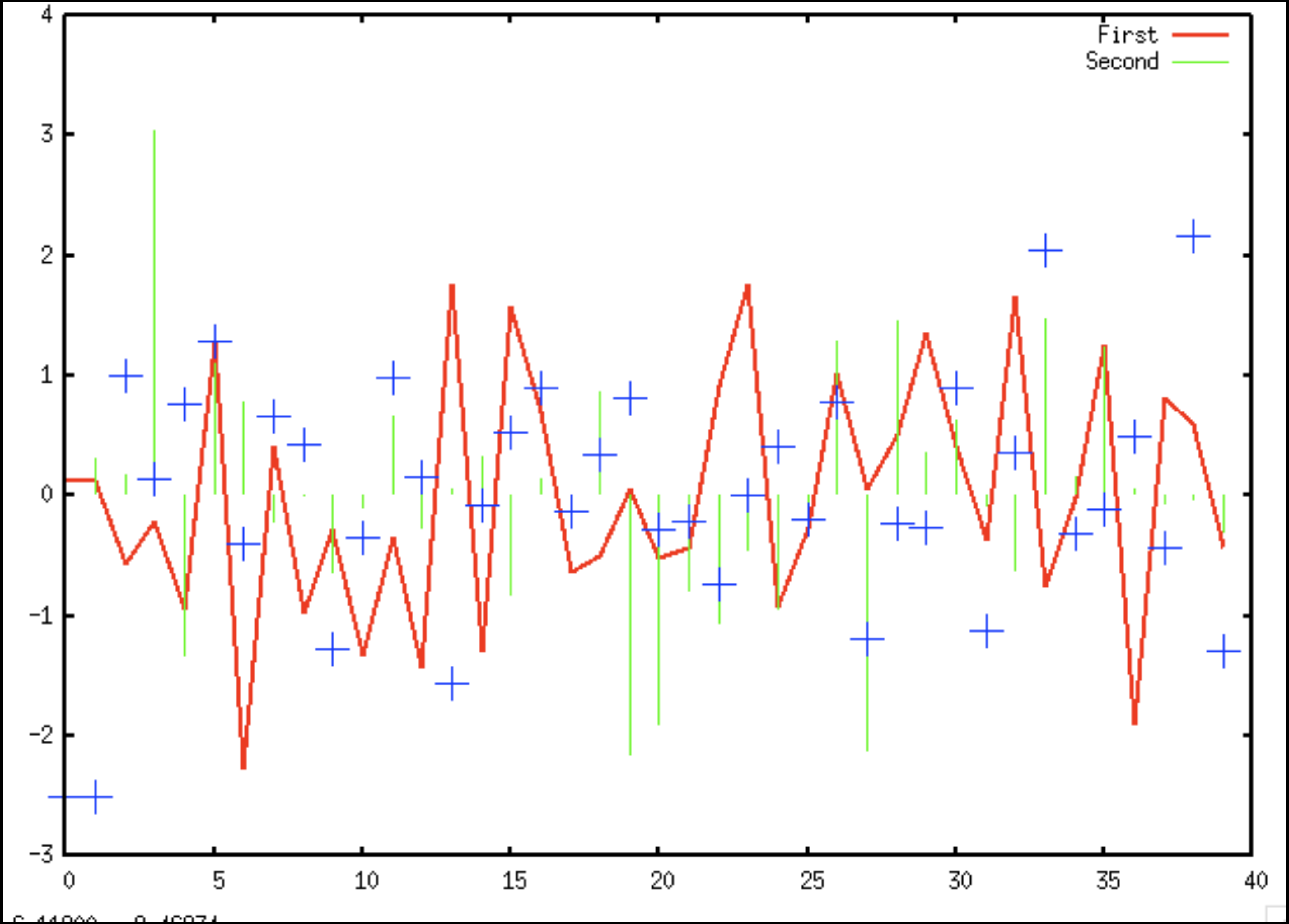
imp impulse (bars)

using 1:2 data in the 2nd column

ps point size

title “title” or notitle

long line



run in commandline

```
$ gnuplot -persist <<< “plot \”file1.txt\” w lines”
```

Displays the result in a window, then exits

```
$ gnuplot <<< 'set term png; set output "out.png"; plot "file1.txt" w lines'
```

Plots into the file “out.png”

in general...

To match a pattern: grep

To work with columns: awk

To replace text: sed

To work only with the first/last lines: head and tail

Iterations on files and numbers can be done with a one-line bash script

sort, uniq, wc, seq, cut and paste can simplify other scripts

And now you have learned the basics of vim and gnuplot

in general...

Now your head is burning with commands and switches!

Just start using these tools step by step, and in a few days you will discover how much time you were wasting by doing things by hand!

Want to learn more? Google! For example, “use awk to filter numbers”

And remember to stick the PDF cheat sheet in front of your desk :)

real life examples

Add the third and fourth columns of a comma separated text if they are positive, only for the two first rows

```
$ head -n 2 file.csv | awk -F "," 
'$2 > 0 && $3 > 0 {print $2+$3}'
```

real life examples

Run a script and check if the results are OK

```
$ for i in *.fa; do
    res_name=`sed "s/\.fa/\.txt/g" <<< "$i"`
    size=`wc -l results/$res_name | awk {'print $1'}`
    if [ "$size" != "500" ]; then
      echo "The size of $i is incorrect";
    fi
 done
```

If you need to go this far... maybe it's OK to use Perl or whatever you're comfortable with

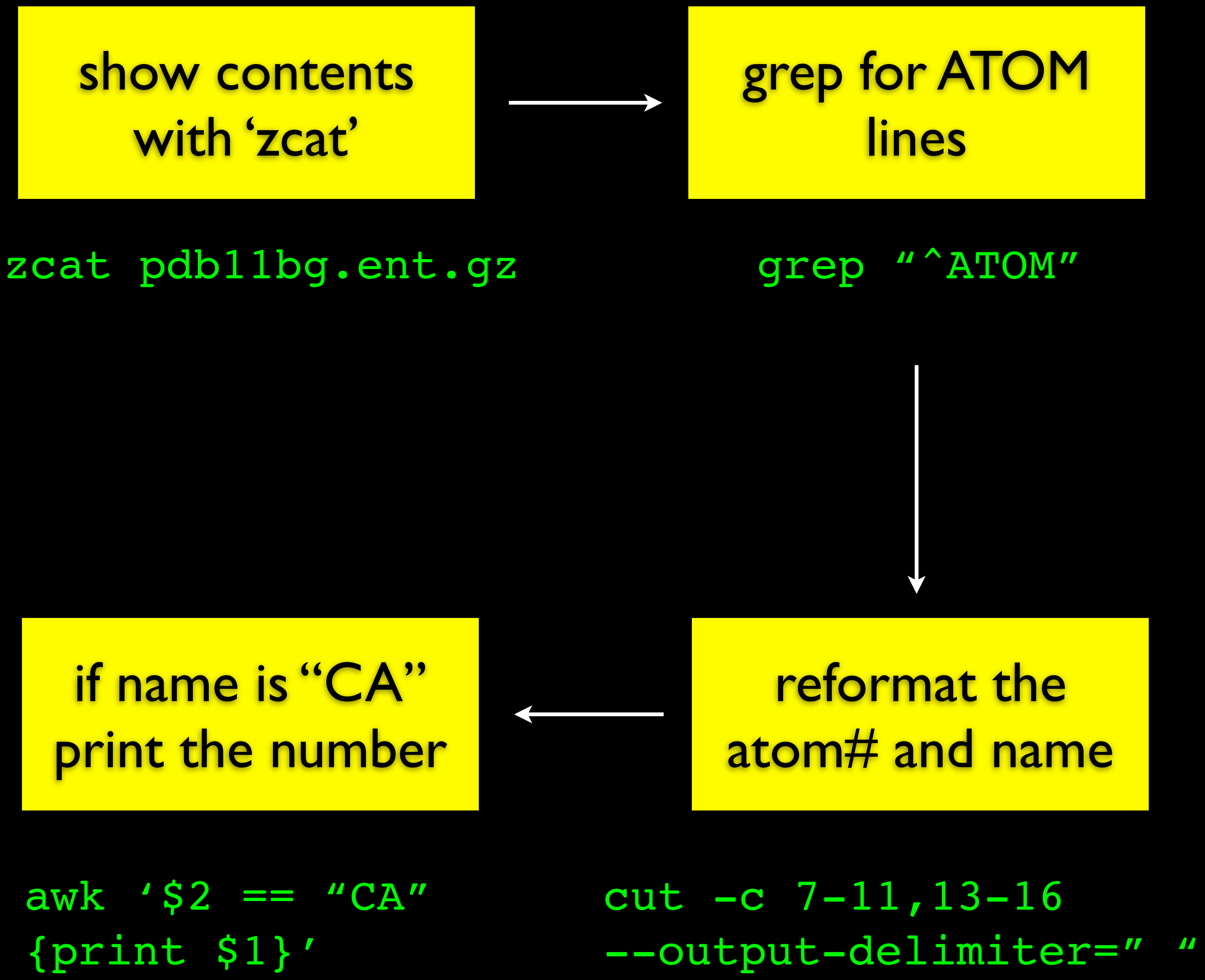
real life examples

Get the atom number of the alpha-carbons from a PDB protein

Problem: ‘awk’ can’t parse byte by byte and the PDB format does not guarantee the presence of a separator (space, comma...)

Solution: Use ‘cut’ to generate our own data format, then parse it with ‘awk’

(Stay cool, if you get this example straigtaway, you probably know more than me!)


show contents with ‘zcat’
grep for ATOM lines
reformat the atom# and name
if name is “CA” print the number

```
zcat pdb1lbg.ent.gz
```

```
grep “^ATOM”
```

```
cut -c 7-11,13-16
--output-delimiter=” “
```

```
awk ‘$2 == “CA”
{print $1}’
```

too complex?

```
$ zcat pdb1lbg.ent.gz | grep "^ATOM" | cut -c
7-11,13-16 --output-delimiter=" " | awk '$2 ==
"CA" {print $1}'

2

11

20

26

31

36

41

...
```

real life examples

Please ask something!

(We’re done, it’s time to ask questions and such)

Thanks to Jordi Camps for his feedback!