Getting to grips with Unix and the Linux family

David Chiappini, Giulio Pasqualetti, Tommaso Redaelli

Torino, International Conference of Physics Students

August 10, 2017

According to the booklet

At this end of this session, you can expect:

- To have an overview of the history of computer science
- To understand the general functioning and similarities of Unix-like systems
- To be able to distinguish the features of different Linux distributions
- To be able to use basic Linux commands
- To know how to build your own operating system
- To hack the NSA
- To produce the worst software bug EVER

~~According to the booklet~~ update

At this end of this session, you can expect:

- ~~To have an overview of the history of computer science~~
- *To understand the general functioning and similarities of Unix-like systems*
- ~~To be able to distinguish the features of different Linux distributions~~
- To be able to use basic Linux commands
- ~~To know how to build your own operating system~~
- ~~To hack the NSA~~
- **To produce the worst software bug EVER**

ICPS 2017 TORINO

A first data analysis with the shell, sed & awk

an interactive workshop

1. at the beginning, there was UNIX...
2. ...then there was GNU
3. getting hands dirty
    - common commands
    - wait till you see piping
4. regular expressions
5. sed
6. awk
7. challenge time

What's UNIX

- Bell Labs was a really cool place to be in the 60s-70s
- UNIX was a OS developed by Bell labs
- they used C, which was also developed there
- UNIX became the *de facto* standard on how to make an OS

ICPS 2017 TORINO

UNIX Philosophy

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

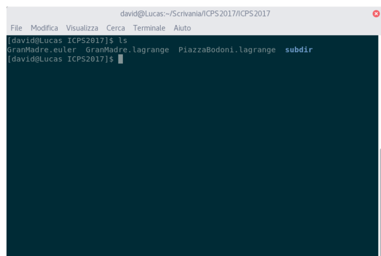
ICPS 2017 TORINO

So what's GNU/Linux?

- GNU's Not Unix
- GNU is a project which rewrote most of UNIX utilities and released them under free software
- GNU/Linux is an OS composed by GNU utilities built on top of a Linux kernel
- Unlike UNIX, GNU and Linux are both Free Software

...BASH?

- Bash (Bourne Again SHell) is a Command line interface
- Bash allows you to use all of the system's utilities and applications and combine them to do more complex tasks
- Most commands will accept a standard input and will print stuff on the standard output



ICPS 2017 TORINO

How to invoke a spell

Every command is invoked using its own name

```
$ ls
GranMadre.euler GranMadre.lagrange
PiazzaBodoni.lagrange subdir
```

You can then add options to change its behaviour (usually consisting of a dash followed by a single letter)

```
$ ls -a
GranMadre.euler GranMadre.lagrange
PiazzaBodoni.lagrange subdir .hiddensubdir
```

ICPS 2017 TORINO

You usually might want (or will have) to add one ore more arguments

```
$ ls subdir
Atwood.euler
```

The options might also have their own arguments

```
$ head -n 1 GranMadre.lagrange
Today I was in Gran Madre church, lost in my thoughts,
when suddenly not far from me, a curious child caught
my attention;
```

Bash pills

Bash is an actual programming language, and thus supports variables, flow control, basic operations.
Bash has no typing and needs no declarations

```
#!/bin/bash
var=0
for i in (5..1); do
echo "Computer exploding in $i s"
sleep 1
done
```

Magic Tricks

Bash also has some magic expressions:

- * the wildcard, which assumes every possible value at the same time
- . is the directory you’re in (working directory)
- .. is always the parent directory of the one you’re in

ICPS 2017 TORINO

Some easy commands

echo prints its argument

```
$ echo "The Lannisters send their regards"
The Lannisters send their regards
```

touch creates a file

```
$ touch afile
```

pwd prints the current (”working”) directory

```
$ pwd
/home/david/Desktop/ICPS2017/playground
```

ls prints the content of a directory

```
$ ls subdir
Atwood.euler
```

cd changes directory

```
$ cd subdir
```

ICPS 2017 TORINO

man: the user's little helper

Argument any command – it will give you exaustive info about that command!

```
LS(1)                          User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort  is  speci-
       fied.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file
```

cat: concatenates two files

Argument

two files which cat will concatenate and print to standard output

Common Options

-n – numbers lines
-b – numbers lines but ignores empty ones
-s – removes repeated blank lines

head and tail

Head and tail respectively print the start or the end of a file.

Argument

The file whose content is to be printed.

Common Options

-n – used to define the number of lines to print.

ICPS 2017 TORINO

sort: sorts lines of the arguments

Argument

The file (with no arguments it reads from standard input).

Common Options

-g – sort by numerical value
-d – alphabetical order

ICPS 2017 TORINO

grep: finds matching lines

Arguments The first argument is a pattern to be matched against, the second argument is a text file. If any line contains the string, it is printed on standard output.

Common Options

-E – accepts extended regular expressions (more on that later)

-f – obtain patterns from a file (given as an argument)

-e – it needs an argument which is a pattern, can be used to match the file against more patterns

and many more, check them out with man grep.

ICPS 2017 TORINO

I/O management

Most utilities will just output stuff on your screen. You can **redirect** this output by using a few symbols:

- > prints the output in a file. Overwrites the file.

  ```
  ls > NewFile.txt
  ```

- » appends the output to a file.

  ```
  ls >> SomeFile.txt
  ```

- < accepts the file on the right as input for the command on the left

  ```
  sort -g < items_to_sort.txt
  ```

- | redirects the output of the left command into the input of the right command

  ```
  ls | head -n 1
  ```

ICPS 2017 TORINO

Regular???? Expressions

Introduction: The Engine

A regular expression ”engine” is a piece of software that can process regular expressions, trying to match the pattern to the given string. Usually, the engine is part of a larger application and you do not access the engine directly. Rather, the application invokes it for you when needed, making sure the right regular expression is applied to the right file or data.

Basic Regular Expressions

- Single literal character: given character a and string s: the engine matches the first occurrence of a in s.

- multiple characters: given regex Euler and string Lagrange writes to Euler:
  - the engine searches for an E, immediately followed by u, then an l and so on.
  - if Eur found: start again
  - Euler and euler are different matches!

ICPS 2017 TORINO

Special Character

There are 12 characters with special meaning:

1. the backslash \
2. the caret ^
3. the dollar sign $
4. the dot .
5. the pipe |
6. the question mark ?
7. the asterisk or star *
8. the plus sign +
9. the opening parenthesis (
10. the closing parenthesis)
11. the opening square bracket [
12. the opening curly brace {

escaping

If you want to use any of these characters as a literal in a regex, you need to escape them with a backslash.

Non printable character

You can use special character sequences to put non-printable characters in your regular expression.

1. the tab \t
2. the carriage return \r
3. the line feed \n

ICPS 2017 TORINO

the dot . and the question mark ?

- . matches any single character
  Exception: line break characters

- ? makes the preceding token in the regular expression optional
  Eg: colou?r matches both colour and color

- You can make several tokens optional by grouping them together using parentheses
  Eg: Nov(ember)? matches Nov and November

ICPS 2017 TORINO

the star * and plus +

- *: match preceding token **zero or more** times
- +: match preceding token **one or more** times

Eg: <[A-Za-z][A-Za-z0-9]*> matches an HTML tag without any attributes:
The angle brackets are literals. First [] matches a letter. 2nd [] matches a letter or digit. * repeats the second []. Because we used *, it's OK if the second character class matches nothing.

character sets []

- Match only **one out of several** characters inside the square brackets
  Eg: if you want to match an a or an e, use [ae]
- A character set matches only a single character
- **Hyphen** inside character set: specify a range of characters.
  Eg: [0-9] matches a single digit between 0 and 9
- ?, * or + after character set: apply to entire character set
  Eg: [0-9]+ can match 837 as well as 222

ICPS 2017 TORINO

metacharacters inside character sets

- in most regex flavors, the only special characters inside a character set are the closing bracket, \, ^, -
- to include them as normal characters: escape them with a backslash (sometimes more than 1)
  Eg: [\\x] matches \ or x

sed

stream editor

sed: stream editor

- parse and transform text
- standard input → transform → standard output
- parsing is line by line
- piping:

```
$ echo "it's a trap" | sed s/ra/ar/
it's a tarp
```

- on file:

```
$ sed 's/ra/ar/' myfile
```

sed substitution

```
$ sed '[addresses]s/pattern/replacement/flag'
```

ICPS 2017 TORINO

playing with substitutions (1)

- substitute in every occurrence in a line:

```
$ sed 's/old/new/g' myfile
```

  without g-flag only first occurrence is substituted!

- substitute only second occurrence on each line:

```
$ sed 's/TAB/>/2' myfile
Column1TABColumn2TABColumn3TABColumn4
Column1TABColumn2>Column3TABColumn4
```

ICPS 2017 TORINO

playing with substitutions (2)

- substitute occurrences only in first line:

```
$ sed -e '1s/old/new/' myfile
```

- substitute occurrences only in last line:

```
$ sed -e '$s/old/new/' myfile
```

- multiple substitutions:

```
$ sed -e 's/old/new/' -e 's/bad/good/' myfile
```

ICPS 2017 TORINO

sed: operations alternative to substitution

- d: delete

  `$ sed '1d' myfile`

  delete the first line of `myfile`

- i: insert

  `$ sed 'N i # t(s) x(m)' myfile`

  insert `# t(s) x(m)` before line `N` in `myfile`

- a: append

  `$ sed 'N a # t(s) x(m)' myfile`

  append `# t(s) x(m)` after line `N` in `myfile`

sed: permanent modification in a file

- Without specific flags sed streams to standard output
- The input file is left unchanged!

- use flag -i to save modification in input file

```
$ sed -i 's/old/new/' newfile
```

ICPS 2017 TORINO

awk

a complete programming language in the shell

structure of awk programs

- text files processing
- files seen as a set of records, by default lines

awk syntax

```
$ awk 'condition { action }' myfile
```

ICPS 2017 TORINO

awk: print

```
$ awk '{ print $1, $3 }' data.txt > filtered-data.txt
```

copies the first and third columns of `data.txt` into `filtered-data.txt`

data.txt

```
5    9.96   -1.8
6    9.95   -2.3
7    9.92   -5.6
9    9.91   -3.2
11   9.85   -4.9
```

filtered-data.txt

```
5 -1.8
6 -2.3
7 -5.6
9 -3.2
11 -4.9
```

ICPS 2017 TORINO

awk: printf (1)

```
printf format, item1, item2, …
```

Example:

```
$ awk '{ printf "%s\n", $1 }' data.txt
```

prints the first column of `data.txt` with a string formatting

awk: printf (2)

```
printf format, item1, item2, …
```

format: %width.precisioncontrol

Control letters:

d	integer
e	scientific notation
f	floating-point notation
g	scientific or floating notation, minimizing number of characters
s	string

Examples:

- `printf "%4.3e", 1950` prints `1.950e+03`
- `printf "%4.3f", 1950` prints `1950.000`

built-in variables

- can be used both in condition and action part
- some examples:
  - **NR**: keeps a current count of the number of input records

    ```
    $ awk 'NR==1 { action }'
    ```

    acts on first line only
  - **FS**: specifies the input field separator

    ```
    $ awk 'BEGIN { FS = ":" }; { print $1,$3 }'
    ```

    `1:2:3` $\rightarrow$ `1 3`
  - **OFS**: specifies the output field separator

    ```
    $ awk 'BEGIN { OFS = ":" }; { print $1,$3 }'
    ```

    `1 2 3` $\rightarrow$ `1:3`

awk: mathematical manipulation

- arithmetic operations between columns are allowed

```
$ awk '{ sum = $2 + $3 + $4 ; avg = sum / 3; print $1, avg }'
data.txt
```

- numerical functions are available:
  - int(x)
  - sqrt(x)
  - sin(x)
  - cos(x)
  - exp(x)
  - log(x)
  - atan2(y, x)
  - rand()

ICPS 2017 TORINO

summary

& challenge

summary

- The GNU utilities and bash can help you automate boring operations on files (like bulk renaming, sorting, merging files)
- sed is a ”little” utility great for repetitive modifications to a file (want to change dividers in a csv, anyone?), or it can be used to parse a file and extract some vital information without having to actually open it – never ever open a 2500 lines file in a notepad again just to check the meadata
- awk is really useful if you want to preprocess a file or gain some preliminary results out of it
- Remember that even most graphical IDEs and professional programs make use of some kind of regular expression, and they are really really powerful!
- Be careful not to use too much of this stuff or you might grow a dependance

challenge time

You are now asked to dirt your hands with the shell

1. get a bash shell
   - if you have GNU/Linux or macOS you already have Bash
   - if you have Windows: the easiest way to get it is installing git: `https://git-scm.com/downloads`
2. download the tarball with the challenge at `http://www.icps2017.it/files/unix-challenge.tar.gz`
3. extract it and start solving the challenges! Look at `README.md`
4. these slides are available at `http://icps2017.it/files/unix-slides.pdf`

Have fun!