Eric A. Zizzi
eric.zizzi@polito.it

Introduction to Linux – Recap and exercises

Outline

1. **Quick Recap**
   - ➢ The directory tree
   - ➢ Basic commands
2. **Manipulating files**
   - ➢ Easy Exercises
   - ➢ Intermediate Exercises
   - ➢ Advanced Exercises
3. **Preview: VMD**
   - ➢ Viewing a PDB file
   - ➢ Tweaking the visualization and rendering

Directory Tree and Command Structure

Basic Command Structure

```
$ command –x <args> -y <args> …<target>
```

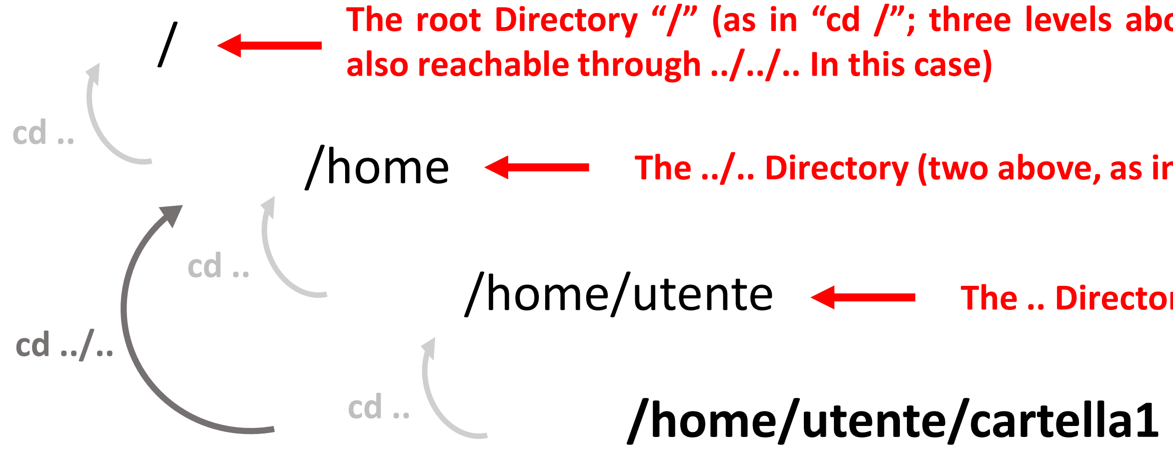
Binary/script. Must be in current folder or in $PATH!

Specific option, with its argument (e.g. –verbose False)

Positional argument, e.g. a target file (if applicable)

Directory Tree and Command Structure

The Directory tree



The root Directory "/" (as in "cd /"; three levels above, also reachable through ../../.. In this case)

The ../.. Directory (two above, as in "cd ../..")

The .. Directory (one above, as in "cd ..")

I am here: The working directory (pwd)

Absolute path	Relative path	Level
/	../../..	Three above
/home	../..	Two above
/home/utente	..	One above
/home/utente/cartella1	**.**	**I am here! $(pwd)**

```
$ command –x <args> -y <args> …<target>
```

Unless explicitly specified, ***command*** will look for files (and create them) **in the current directory**. Different locations must be specified!

Recap of common commands (remember!)

Command name	Use	Notes	Example
ls	List files in current directory	*Can take directory to list as argument! Can format list with options*	***ls -lah /home/utente***
cp	Copies files and folders (with –r)	*Requires source and then destination*	***cp file1 ../cartella2***
mv	Moves files and folders (with –r) or renames them	*Renaming a file is equivalent to moving it into the same directory with a different name!*	***mv nomevecchio nomenuovo***
cd	Change directory; moves to home (~) as a default	*Can use relative or absolute paths. Cannot cd to directories which don't exist (must mkdir them first)*	***cd ../../../cartella3***
mkdir	Creates directory with name given as argument in current directory	*Argument can be absolute path to folder to create: you can create folders in locations different from where you currently are!*	***mkdir /home/nuovacartella***
echo	Prints text to terminal	*Can be piped to add text to a text file!*	***echo "Ciaociao" >file.txt***
export	Exports variable to current shell environment	*Variable can be accessed only within the same shell! If you close and re-open the terminal, the variable will be lost!*	***export pippo***
touch	Creates an empty file; name specified as argument	*Might seem useless, but comes in handy many times!*	***touch fermacarte.fc***
cat	Concatenates file content to stdout	*In other words, print content of file to terminal*	***cat readme.txt***
nano	Command-line text editor	*Uses key combinations instead of usual GUI buttons while editing: Ctrl-X to exit, Ctrl-O to save, etc…*	***nano ~/.bashrc***
head/tail	Prints first/last 10 lines of a file to STDOUT	*You can use the option -n to specify how many lines to print!*	***head -n 100 testo.txt***
grep	Searches file for lines containing a pattern	*Spend some time exporing all the different options of grep!*	***grep "query-text" file.txt***
uniq	Filters out adjacent repeated lines of a text file	*Detects duplicates only if adjacent! Might need to sort the lines first!*	***uniq text.txt***
sort	Sorts contents of a text file line-by-line	*Sorts numerically then alphabetically (lowercase first). See options*	***sort input.txt >sorted.txt***
cut	Extracts specific parts of each line of a text file	*Useful for example to print first n characters of each line, or print specific column*	***cut -d " " -f 1 columns.txt***
wc	Word count. Used for counting in text files.	*Has many options; useful for counting lines with -l option*	***cat text.txt | wc -l***
awk	Scripting language to manipulate files	*Incredibily powerful, but slow learning curve! We'll see examples today…*	***awk '{print $1,$2}' file.txt***

Clever hints!

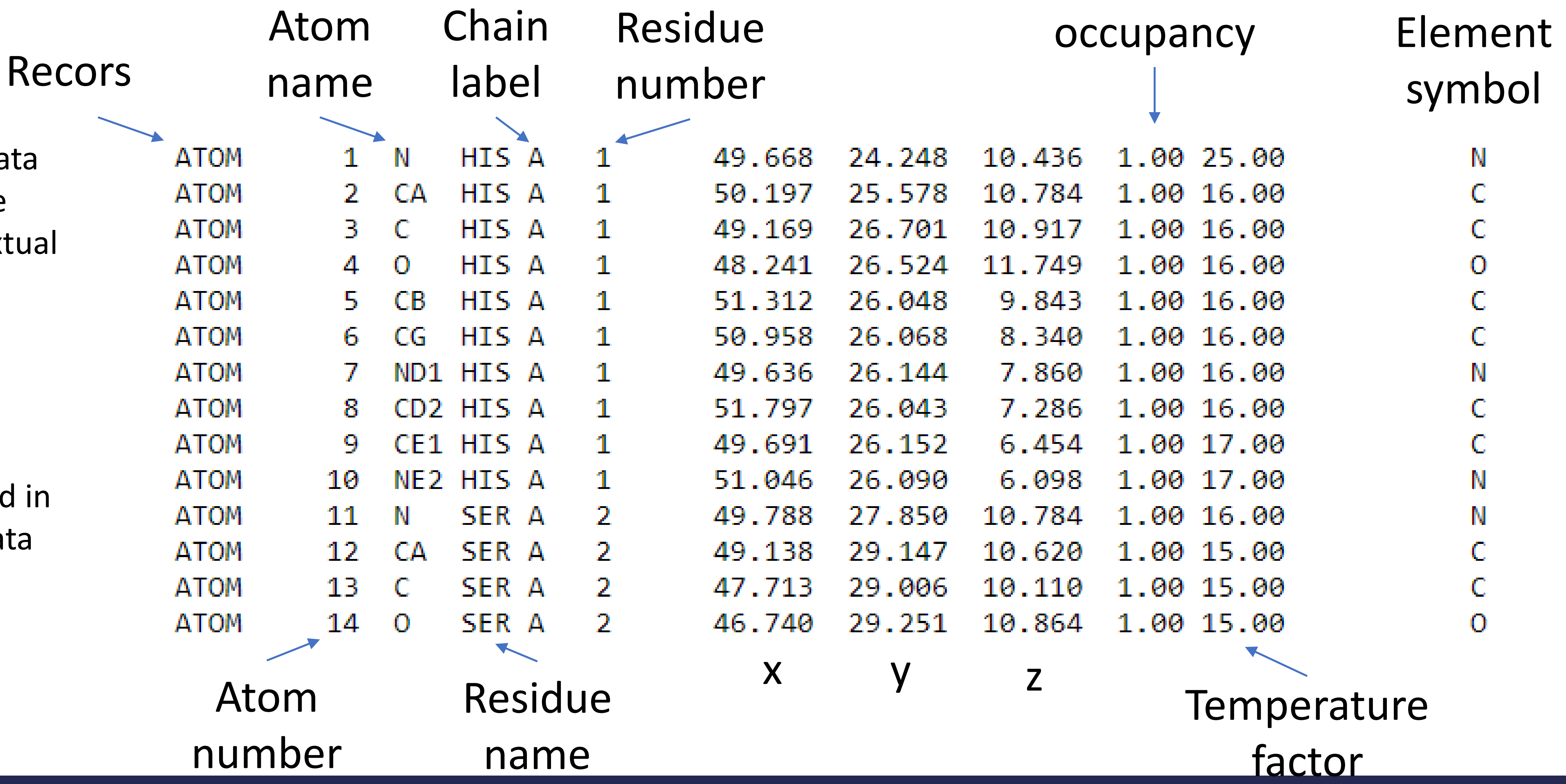
Here are some very useful tips and caveats

- Hidden files in Linux begin with a **.** (example: .bashrc)
- Different terminal outputs exist: the two main ones are **STDOUT** and **STDERR**
- **/dev/null** is a virtual device that does not retain any file or information passed to it (everything deleted and not present in memory)
- **>** at the end of a command writes the STDOUT to a file (and overwrites it!)
- **>>** at the end of a command concatenates the STDOUT to a file, appending it to the end of existing files
- You can pipe both **STDOUT** and **STDERR** to the same file by appending **>log.txt 2>&1** at the end of the command!
- **rm** and its different flavours do not have a trash folder! Removed files are lost forever!
- Characters between backticks **`** … **`** are executed as commands (in a different subshell!)
- Most commands have default options you should know, so that you know their behaviour if you don't specify any arguments! (See for example **cd**)
- Use tab completion to auto-complete commands in your shell!

A PDB file

https://www.rcsb.org/

- The Protein Data Bank (pdb) file format is a textual file format describing the **three-dimensional structures of molecules** held in the Protein Data Bank.



```
ATOM      1  N   HIS A   1      49.668  24.248  10.436  1.00 25.00           N
ATOM      2  CA  HIS A   1      50.197  25.578  10.784  1.00 16.00           C
ATOM      3  C   HIS A   1      49.169  26.701  10.917  1.00 16.00           C
ATOM      4  O   HIS A   1      48.241  26.524  11.749  1.00 16.00           O
ATOM      5  CB  HIS A   1      51.312  26.048   9.843  1.00 16.00           C
ATOM      6  CG  HIS A   1      50.958  26.068   8.340  1.00 16.00           C
ATOM      7  ND1 HIS A   1      49.636  26.144   7.860  1.00 16.00           N
ATOM      8  CD2 HIS A   1      51.797  26.043   7.286  1.00 16.00           C
ATOM      9  CE1 HIS A   1      49.691  26.152   6.454  1.00 17.00           C
ATOM     10  NE2 HIS A   1      51.046  26.090   6.098  1.00 17.00           N
ATOM     11  N   SER A   2      49.788  27.850  10.784  1.00 16.00           N
ATOM     12  CA  SER A   2      49.138  29.147  10.620  1.00 15.00           C
ATOM     13  C   SER A   2      47.713  29.006  10.110  1.00 15.00           C
ATOM     14  O   SER A   2      46.740  29.251  10.864  1.00 15.00           O
```

PDB file format

Record Format

COLUMNS	DATA TYPE	FIELD	DEFINITION
1 - 6	Record name	"ATOM "	
7 - 11	Integer	serial	Atom serial number.
13 - 16	Atom	name	Atom name.
17	Character	altLoc	Alternate location indicator.
18 - 20	Residue name	resName	Residue name.
22	Character	chainID	Chain identifier.
23 - 26	Integer	resSeq	Residue sequence number.
27	AChar	iCode	Code for insertion of residues.
31 - 38	Real(8.3)	x	Orthogonal coordinates for X in Angstroms.
39 - 46	Real(8.3)	y	Orthogonal coordinates for Y in Angstroms.
47 - 54	Real(8.3)	z	Orthogonal coordinates for Z in Angstroms.
55 - 60	Real(6.2)	occupancy	Occupancy.
61 - 66	Real(6.2)	tempFactor	Temperature factor.
77 - 78	LString(2)	element	Element symbol, right-justified.
79 - 80	LString(2)	charge	Charge on the atom.

https://cupnet.net/pdb-format/

Exercise 1.1 (easy)

Create a folder in your home called “ese01” and move into it. Print the folder path and save it in a variable called **pathx**. Print the variable on your screen.

Hints: echo, ….

Exercise 1.1 (solution)

```
$ mkdir /home/studente/ese01
$ cd ~/ese01
$ pathx=$(pwd)
$ echo $pathx
```

Exercise 1.2 (easy)

Create a file called file01.txt; write the pathx variable into it; finally, print the file to the terminal

Hints: echo, touch,…

Exercise 1.2 (solution)

```
$ touch file01.txt
$ echo $pathx >>file01.txt
$ cat file01.txt
```

Question: are there other ways to solve the exercise? Let’s discuss!

Exercise 1.3 (easy)

Copy the 1yzb.pdb file into the ese01 folder you just created. Print all the 1yzb.pdb file to the terminal window. Now, clear the command window and print the first 50 lines of the 1yzb.pdb file; copy these first 50 lines into a file named first50.pdb. Check that this succeeded by counting the lines of first50.pdb.

Hints: cp, clear, cat, head, wc

Exercise 1.3 (solution)

```
$ cd ~/ese01
$ cp ~/Desktop/1yzb.pdb .
$ cat 1yzb.pdb
$ clear
$ head -n 50 1yzb.pdb
$ head -n 50 1yzb.pdb >first50.pdb
$ cat first50.pdb | wc -l
```

Question: are there other ways to solve the exercise? Let’s discuss!

Exercise 2.1 (intermediate)

- Extract only the ATOM lines from 1yzb.pdb and save them to a file named atoms.pdb;
- Extract only the first 10 ATOM lines of 1yzb.pdb, and only the last 10 ATOMS of 1yzb.pdb: save these into first10.pdb and last10.pdb files respectively;
- Extract only ATOM lines from 10 to 20 from 1yzb.pdb file;

Hints: grep, head, tail

Exercise 2.1 (solution)

```
$ cat 1yzb.pdb | grep ATOM >atoms.pdb
$ head -n 10 atoms.pdb >first10.pdb
$ tail -n 10 atoms.pdb >last10.pdb
$ tail -n +10 atoms.pdb | head -n 11
```

Question: are there other ways to solve the exercise? Let’s discuss!

Exercise 2.2 (intermediate)

- Create a new file named extracted.pdb consisting of ATOMs 1-10, 100-110 and the last 10 ATOMS from 1yzb.pdb
- Extract only ATOMS with x lower than 1.5, and save them into x_below.pdb
- Extract only x y z coordinates and atom numbers from ATOMS in the 1yzb.pdb file; save this data to simple.pdb

Hints: cat, grep, head, tail, cut, awk

Exercise 2.2 (solution)

```
$ cat 1yzb.pdb | grep ATOM >atoms.pdb
$ head -n 10 atoms.pdb >extracted.pdb
$ tail -n +100 atoms.pdb | head -n 11 >>extracted.pdb
$ tail -n 10 atoms.pdb >>extracted.pdb

$ awk ’$7<1.5 {print $0}’ atoms.pdb >x_below.pdb

$ awk ‘{print $2, $7, $8, $9}’ atoms.pdb >simple.pdb
```

Question: are there other ways to solve the exercise? Let’s discuss!

Exercise 3.1 (advanced)

- Count how many ALA residues are contained in 1yzb.pdb
- Count how many trajectory steps are in 1yzb file (every step is separated by the word MODEL)
- Write a file sequence.txt with a number sequence from 1 to 1000

Hints: cat, grep, wc, uniq, sort, seq

Exercise 3.1 (solution)

```
$ grep ALA atoms.pdb| grep CA | sort | uniq -w 20 | wc -l
$ grep -v REMARK 1yzb.pdb | grep MODEL | wc -l
$ seq 1 1000 >sequence.txt
```

Note the convoluted one-liner in the first step! Do you have any solution which is more elegant? Are there more ways to solve this problem?
Why the first grep in the second line? Any ideas?

Scripting bash

Bash can be scripted as C, python, js or other languages and has tools like **conditional** and **cycles**

IF

```
If <test>
then
 <dosomtehing>
else
<dosomething>
fi
```

FOR

```
for <variable> in <list>
do
 <dosomtehing>
done
```

Example

```
$ n=10
$ if [$n -gt 5]
$ then
$ echo “ your number is greater
than 5”
$ else
$ echo “your number is lower or
equal to 5”
$ fi
```

Example

```
$ for i in $(ls)
$ do
$ echo $i
$ done
```

Commonly used operators

In bash test operator can be in either one of this format:

```
$ if <test>
$ If [<test>]
$ If [[<test>]]
```

Some (not all) of possible tests:	
-n VAR	True if the length of VAR is greater than zero.
-z VAR	True if the VAR is empty.
STRING1 = STRING2	True if STRING1 and STRING2 are equal
STRING1 != STRING2	True if STRING1 and STRING2 are not equal.
INTEGER1 -eq INTEGER2	True if INTEGER1 is equal to INTEGER2
INTEGER1 -lt INTEGER2	True if INTEGER1 is less than INTEGER2.
INTEGER1 -ge INTEGER2	True is equal or greater than INTEGER2
INTEGER1 -le INTEGER2	True is equal or less than INTEGER2
-h FILE	True if the FILE exists and is a symbolic link
-d FILE	True if the FILE exists and is a directory.
-f FILE	True if the FILE exists and is a regular file (e.g. not a directory).
-e FILE	True if the FILE exists and is a file, regardless of type (node, directory, socket, etc.).

Scripting bash

Bash scripts must start with #!/bin/bash, only for the first raw the symbol # is not seen as a comment, then it will be interpreted as a comment and the line will be ignored

currentfolder.sh

```
#!/bin/bash

#questo è un commento
#non viene eseguito dal terminale
echo "the current folder is $pwd"
#concludo lo script
exit 0
```

To execute the script I can either lunch the script as an argument of bash

```
$ bash nameofthescript.sh
```

or make it executable and lunch it as a program

```
$sudo chmod 777 nameofthescript.sh
$./nameofthescript.sh
```

NB: in the latter case it is necessary to explicit the interpreter in the first line of the script, i.e. "#!/bin/bash"

Exercise 4.1(Scripting)

Create a script named “proteinstats.sh” that:

- reads all the “.pdb” files in a folder
- For each “.pdb” file, add to a file named “stat.csv” a row with the formatting “<filename>,<residues>”
- Update a file called “maxres.stat” with a single row

  “ <filename> has the highest number of residues equal to <residues>”

  , with the name of the .pdb file owning the highest number of residues and the total number of residues.

Hints for, if, >, >>

Example of the expected result:

```
$ bash proteinstast.sh
$ cat stat.csv
TAS1R1.pdb,305
TAS1R2.pdb,398
...
TAS2R3.pdb,320
$ cat maxrest.stat
TAS1R2 has the highest number of residues equal to 398
```

Viewing PDB files with VMD

(Practical demo)

Questions?