PCCP/M2 Molecular Simulation

4TCH914U

Practice

J-C. Soetens

`jean-christophe.soetens@.u-bordeaux.fr`

Academic year 2024-2025

University of Bordeaux

P0: Numerical environment

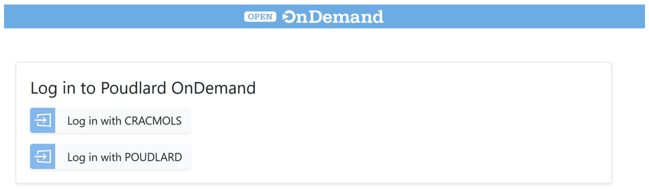
All practical work will take place on the POUDLARD server dedicated to teaching at the regional computing center (MCIA : Mésocentre de Calcul Intensif Aquitain) hosted on the campus of the University of Bordeaux.

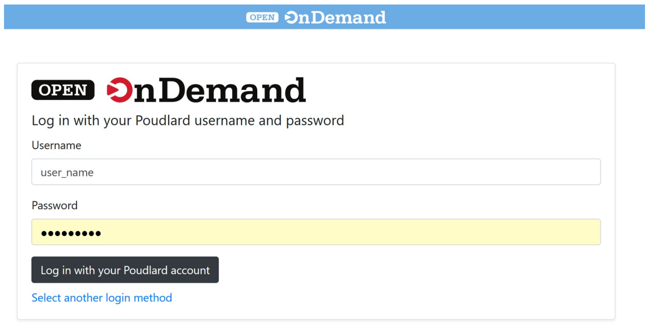
Address of POUDLARD : `poudlard.mcia.fr`

The cluster can be accessed in two different ways (mainly method 2 in our practical sessions) :

1) To access the cluster's front-end nodes from a Unix session:
   ssh user_name@poudlard.mcia.fr

2) Using the web service
   `https://chemins-de-traverse.mcia.fr`
   allows users to access POUDLARD from a web browser (firefox, chrome, etc.)

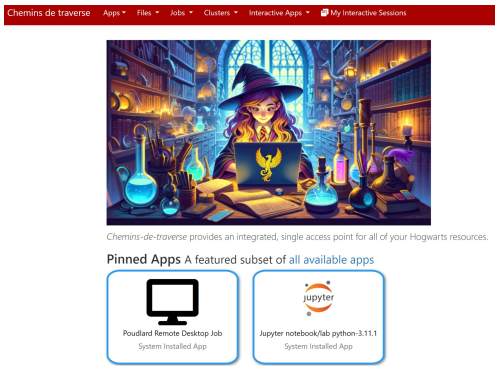



OPEN OnDemand
OPEN OnDemand
Log in with your Poudlard username and password

Username: user_name

Password: •••••••••

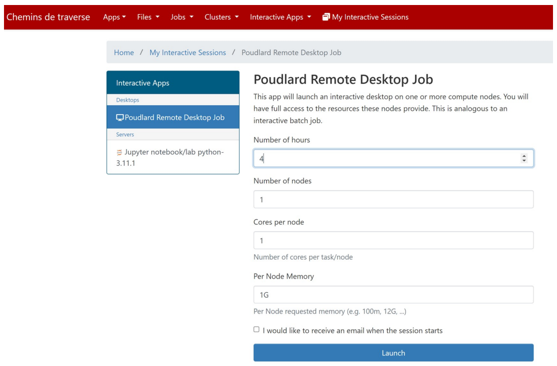
Log in with your Poudlard account
Select another login method



Chemins-de-traverse provides an integrated, single access point for all of your Hogwarts resources.

Pinned Apps A featured subset of all available apps

Poudlard Remote Desktop Job
System Installed App
jupyter
Jupyter notebook/lab python-3.11.1
System Installed App


Chemins de traverse
Apps
Files
Jobs
Clusters
Interactive Apps
My Interactive Sessions
Home / My Interactive Sessions / Poudlard Remote Desktop Job
Interactive Apps
Desktops
Poudlard Remote Desktop Job
Servers
Jupyter notebook/lab python-3.11.1
Poudlard Remote Desktop Job

This app will launch an interactive desktop on one or more compute nodes. You will have full access to the resources these nodes provide. This is analogous to an interactive batch job.

Number of hours: 4

Number of nodes: 1

Cores per node: 1

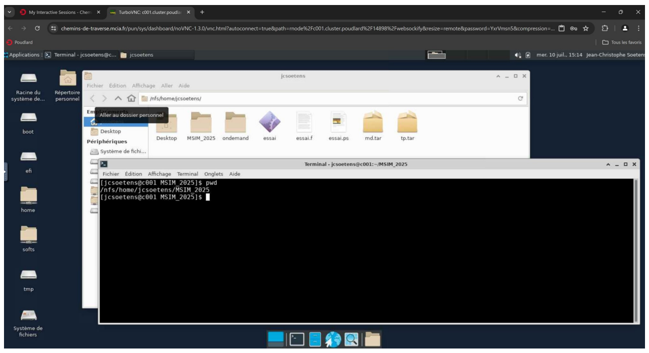
Number of cores per task/node

Per Node Memory: 1G

Per Node requested memory (e.g. 100m, 12G, ...)

☐ I would like to receive an email when the session starts

The result is a multi-window graphical environment :


My Interactive Sessions - Chem
TurboVNC: c001.cluster.poudlar
chemins-de-traverse.mcia.fr/pun/sys/dashboard/noVNC-1.3.0/vnc.html?autoconnect=true&path=rnode%2Fc001.cluster.poudlard%2F14898%2Fwebsockify&resize=remote&password=YxrVmsn5&compression=...
Poudlard
Tous les favoris
Applications
Terminal - jcsoetens@c...
jcsoetens
mer. 10 juil., 15:14 Jean-Christophe Soetens
Racine du système de...
Répertoire personnel
boot
efi
home
softs
tmp
Système de fichiers
jcsoetens
Fichier Édition Affichage Aller Aide
/nfs/home/jcsoetens/
Aller au dossier personnel
Desktop
Périphériques
Système de fichi...
Desktop
MSIM_2025
ondemand
essai
essai.f
essai.ps
md.tar
tp.tar
Terminal - jcsoetens@c001:~/MSIM_2025
Fichier Édition Affichage Terminal Onglets Aide

```
[jcsoetens@c001 MSIM_2025]$ pwd
/nfs/home/jcsoetens/MSIM_2025
[jcsoetens@c001 MSIM_2025]$ 
```

The Poudlard cluster is made up of several groups of accessible machines :

2 front-end interactive nodes: machines to which the user connects to develop and launch their work, manipulate their data and results.

32 Apollo2000 compute nodes: these are the machines on which the jobs are executed (via the Job Manager). These are machines c001 to c032, each having 32 AMD cores per node and 250 GB of RAM.

To know more :
`https://gazette-du-sorcier.mcia.fr/projects/poudlard/wiki`

The Poudlard's operating system is `Unix` :
To know more : see **Annex-A) ABC of Unix** of this document.

Main softwares and utilities used in this course

– Text editors : `Mousepad, vi`

– Postscript editor : `evince`

– Fortran compiler : `gfortran`
See also **Annex-B) Start programming in** Fortran of this document.

– Command-line driven graphing utility : `gnuplot`

– Electronic structure modeling :
`Gaussian 16` (& graphical interface `Gaussview 6`)
`Ampac` (& graphical interface `Agui`)

– Molecular Dynamics simulation packages :
`MDpol` (home programm)
`Lammps` (Large-scale Atomic/Molecular Massively Parallel Simulator.)

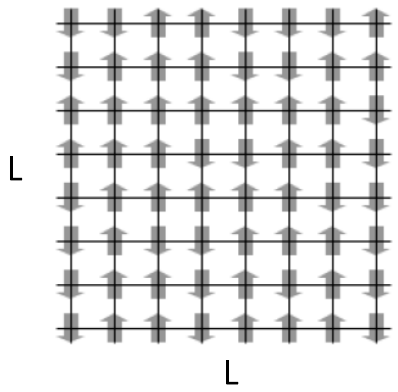
P1: From statistical mechanics to simulation methods.

P1-A) : Ising model

The objective of this part is to use a simple model to illustrate the basic concepts of statistical thermodynamics. The Ising Model is thus used to show the limits of a comprehensive approach ("brute force" approach) and to understand how the simulation allows to push these limits.

Ising model (Ernst Ising 1900-1998)

Example of lattice 8 x 8



Potential energy model :

⬆ $S_i = 1$ ⬇ $S_i = -1$

$$U_{total} = -\epsilon \sum_{neighbours} s_i s_j$$

Partition function :

$$Z = \sum_{conf} e^{-U_i/kT}$$

Computation of a property :

$$< A >= \frac{\sum_{conf} A_i e^{-U_i/kT}}{\sum_{conf} e^{-U_i/kT}}$$

Work to do

1) "Brut force" approach : exact results through the partition function

Become familiar with the program and the system

To do : L = 2 and different kT values

kT	1	2	8	20	100	∞
ELN						
$< E >$						

Interpretation:
- Does the number of energy levels change with kT ?
- Does the energy of each level change with kT ?
- What is the weights of the levels when $kT \to \infty$?

To do : L = 4 and kT=1, 2, 4, 8, 10...
Observe the evolution of the weight of the two lowest energy levels as a function of kT ?

Find the limits

Questions :
- How the computation time evolves according to L ?
- What is the maximum value of L (Lmax) feasible (from computer time point of view) during our afternoon session ?
- What would be the computation time for a system of size Lmax + 1 ?

The command line ”time ./ising” allow you to know the computer time cost of a calculation.

To do : kT = 4 and different L values

L	Total NOS	computer time (appropriate unit)
2		
3		
4		
5		
6		
7		
...		

2) Use of the Monte Carlo method and comparisons with the "Brut force" approach

- Test whether the Monte Carlo method can reproduce the previous results.

To do : L=3, kT =4 and different numbers of MC steps

What is the number of MC steps necessary to reproduce the 'exact result' with less than 1 % of error on the average energy ?

To do : L=4, kT =4 and different numbers of MC steps

What is the number of MC steps necessary to reproduce the 'exact result' with less than 1 % of error on the average energy ?

To do : L=5, kT =8 and different numbers of MC steps

- Find the number of MC steps necessary to reproduce the 'exact result' with less than 1 % of error on the average energy.

- Compare the computation time of the two methods.

3) To infinity... and beyond !

To do : treat the case $(Lmax + 1)$ using the MC method solely...

P1-B) : the Boltzmann distribution

P2: Molecular Dynamics simulations : how it works ?

The objective of this part is to perform first short molecular dynamics simulations.
The directory `P2` contains the following files :

- `MD.dat` : conditions of the MD simulation.
- `bjh.dat` : informations on simulated system.
- `bjh.ff.dat` : informations on simulated system.
- `bjh.conf.dat` : starting configuration.
- `rdf.dat` : control of radial distribution functions.
- `resume.g` : gnuplot script.

Work to do :

1) Note the conditions of the simulation and the nature of the system in editing the file `MD.dat` and `bjh.dat`: thermodynamics conditions, time step, etc.

- What system will you simulate?
- What is the length of the simulation ?

2) Run the simulation using the command `"mdstart MD.dat"` and observe the creation of new files.

- What are the final and average temperatures of the simulation ?
- Same question for the potential energy ?
- What is the CPU computation time (in seconds) of the simulation? (This information appears in the file `nohup.out`).
- Run the command `"gnuplot resume.g"` and view the postscript file produced using the command `"evince resume.ps"`. How to interpret the variations of the different energies ? What information do you get from the examination of the radial distribution functions ?

3) The file `traj.xyz` contains the positions of the atoms recorded every 50 time step, i.e. the trajectory of the system. Visualize this trajectory using the software `vmd`.

- Observe the nouvements of each molecule. One of them has a dynamic radically different from the other .... Which one ? Is there an obvious reason ?

P3: Molecular Dynamics simulations : first applications.

P3-A) Simulation of liquid water.

- Read the files present in the directory `P3A` and note all relevant informations on the simulation.
- Calculate the density of the system based on the number of molecules and the size of the simulation box.
- Start the calculation (command `"mdstart MD.dat"`) and wait the end of the simulation... Compare the CPU time of the simulation with the one observed in the previous part.
- Run the command `"gnuplot resume.g"` and view the postscript file produced using the command `"evince resume.ps"`. Observe the change of the internal energy versus time. What may be the reason for such a change ?
- Interpret the radial distribution functions.
- Display some configurations (software `vmd`).

P3-B) Simulation of an aqueous ionic solution.

- Read the files present in the directory `P3B` and note all relevant information on the simulation.
- Calculate the molar concentrations of ions Na^+ and Cl^-.
- Start the calculation (command `"mdstart MD.dat"`).
- Run the command `"gnuplot resume.g "` and view the postscript file produced using the `"evince resume.ps"`.
- Explain the differences between the potential energies average of the different species.
- Observe the radial distribution functions. How to justify the noisy appearance of the water-ion and ion-ion functions compared to those of water-water ?
- How are the orientations of the water molecules around the ions ? Check your analysis by viewing some configurations.
- Conclude on the solvation of NaCl in water.

P4: Molecular interactions

P4-A) ab initio calculations of potential derived charges.

The objective of this part is to use standard and simple ab initio calculations to get informations on electrostatic properties of simple molecules.

Work to do

- Use the Gaussian09 program to complete the tables on the following two pages.

Electrostatic properties of the water molecule

	Gaussian keywords		
	#p HF/3-21G opt=Z-matrix gfinput IOP(6/7=3)	**#p HF/3-21G pop=ChelpG gfinput IOP(6/7=3)**	**#p HF/3-21G pop=(ChelpG,dipole) gfinput IOP(6/7=3)**
OH bond (A)		=	=
HOH angle (degrees)		=	=
Energy (Hartree)		=	=
SCF dipole (Debye)		=	=
Partial charges model	**Mulliken**	**ChelpG**	**ChelpG (constrained)**
qO (e)			
qH (e)			
corresponding dipole (Debye)			

Electrostatic properties of the water molecule

	Gaussian keywords		
	#p HF/6-31G opt=Z-matrix gfinput IOP(6/7=3)**	**#p HF/6-31G** pop=ChelpG gfinput IOP(6/7=3)**	**#p HF/6-31G** pop=(ChelpG,dipole) gfinput IOP(6/7=3)**
OH bond (A)		=	=
HOH angle (degrees)		=	=
Energy (Hartree)		=	=
SCF dipole (Debye)		=	=
Partial charges model	**Mulliken**	**ChelpG**	**ChelpG (constrained)**
qO (e)			
qH (e)			
corresponding dipole (Debye)			

P4-B) Exploration of a PES and fit of an intermolecular interaction model.

The objective of this part is to fit a simple interaction model for the HF dimer.

Simple model in the present case means :
- rigid model.
- partial charges on atoms to describe the electrostatic contribution to the interaction potential.
- one Lennard-Jones site at the middle of the bond to describe the repulsion and dispersion contributions to the interaction potential.

Preliminary questions :

- write the mathematical expression corresponding to this interaction model.
- how many unknown parameters ?
- what is the dimensionality of the potential energy surface of the HF dimer ?

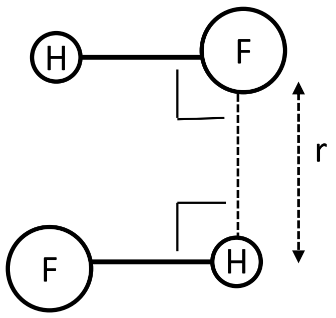
Work to do on the monomer HF

*all calculation will be done using Gaussian09 at the HF/6-31G** level.*

- find the optimized geometry of the HF molecule .
- find the total energy of the HF molecule.
- find the partial charges on H and F using the ChelpG method.

Work to do on the dimer HF

- use Gaussian09 with the "scan" keyword to explore the following dimer planar geometry ($1.8 < r < 8.0\ A$).



- edit the gnuplot script file `"scan.g"` and follow the instructions to fit the parameters of the proposed interaction potential model.

If you have time, you can do the same job but taking into account the basis set superposition error (BSSE).

An example of PES of HF dimer can be found in J. Chem. Phys. 1786, **88**, *1988.*

P5: To go further...

P5-A) Read, analyze, summarize and present a scientific article

Rules : you are free to search for an article on a topic that interests you but obviously connected to this course : molecular simulation (MD or MC) on condensed phases described at the atomic level (pure liquids, mixtures, ionic solutions, unterfacial systems, solids, etc). I await your proposals for validation and do not hesitate to interact with me throughout this exercise.

I remind also you that these presentations are graded and will contribute to the overall score with a coefficient of 0.15. Beyond the quality of the presentations (slides, talk, answers to questions), I will also take into account the participation of everyone during the discussion by the quality of your comments and/or questions to your colleagues.

Input : a scientific article on a topic that interests you and obviously connected to this course ! After approval, you are free to present any paper about molecular simulation you like.

Output : a powerpoint file (around five slides) and a ten-minutes talk in front of the other students.

P5-B) Programming in FORTRAN to develop your own tools

In concrete terms, such a project consists of studying theoretically the problem to propose an algorithm and define the inputs / outputs (including graphics), program it, test it, and present it to colleagues.

Below are some examples of possible programming projects. This list is open for discussion according to your level and your areas of interest. **This work will be done alone or in pairs.** If necessary, I will help you to choose your project but in any case we will have an in-depth discussion on the objectives. I will also help you throughout the project by providing you additional informations.

Bellow some propositions of programming projects :
Programming level : beginner *, advanced **, expert ***, virtuoso ****

- **A)* Random walk in 1D (and maybe in 2D **).**
  The random walk consists of a succession of random steps. It is a famous and ideal system to understand what is a simulation, the sampling problem and.... an important property in physics.

  Work to do :
  - find in the literature what is exactly this model of random walk.
  - think about inputs (data) and outputs (results).
  - write a code implementing this model.
  - calculate various relevant properties and present the results graphically.

- **B)** Analysis of a MD trajectory**.
  Given a trajectory obtained by a MD simulation, write a code to read this trajectory and then calculate some properties. In particular, it is a water / methanol mixture and you will focus on the characterization of the hydrogen bonds in order to calculate various statistics.

  Work to do :

  - learn more in the literature on how to characterize hydrogen bonds.
  - write a code to read the trajectory file.
  - program the calculation of some properties related to hydrogen bonds.
  - present some of the results graphically.

- **C)** Calculation of the volume of a molecule**.
  Starting from a molecular geometry calculated using any software (Gaussian, Ampac...), calculate the volume of a molecule represented by van der Waals spheres (you will use the σ parmeters of the OPLS-AA force field).

  Work to do :

  - learn more in the literature on how to calculate the volume of secant spheres .
  - find an algorithm to do this calculation of volume.
  - think about inputs (data) and outputs (results).
  - write a code implementing this algorithm.
  - calculate the volume of various molecules and present the results graphically.

- **D)*** Monte-Carlo simulation of a pure Lennard-Jones fluid**.
  On the steps of M. Rahman !
  (Physical Review, 1964 : liquid Argon $T = 87$ K, $d = 1.38$, $\sigma = 3.4$ A,$\epsilon = 1$ kJ/mol).

  Work to do :

  - think about inputs (data) and outputs (results).
  - write a code implementing the Monte Carlo algorithm to sample such a system.
  - implement the calculation of the rate of accepted/rejected moves.
  - implement the calculation of the average energy.
  - implement the calculation of the structure of the fluid and compare with the result of 1954 (Rahman, Phys. Rev. **136**, 405 (1964)).

- **E) Initial configuration for a simulation** .
  Whatever the simulation method (MD, MC ...) it is necessary to create an initial configuration. In the case of the study of a disordered condensed phase, it will generally be a cubic box respecting a given density (and composition in case of mixtures).

  This project consists of writing a program generating an initial configuration. You can follow the following progress :

  - (difficulty *) system of atoms (van der Waals spheres).
  - (difficulty *) molecular liquid (small molecule such as HF, H2O, CH4 ...).
  - (difficulty **) mixtures of atoms of several types (mixtures of van der Waals spheres).

  (difficulty ***) It is also necessary to ensure that the energy of the configuration is not aberrant, which could put the simulation in failure after few steps. For example, you can use the Monte Carlo method and a generic force field to relax the system.

  (difficulty ****) At this point you should be able to write a general program capable of generating any configuration.

- **F)**** Monte-Carlo simulation of a binary mixture** .
  This project can be considered if project D is successful and robust. It consists of studying a mixture of two species and may help to understand the miscibility of two species (by varying the ratios of the Lennard-Jones parameters ϵ_A/ϵ_B and $\sigma_A = \sigma_B$.)

- **G) Any (feasible) topic that interests you. To be discussed...**

Annex A ABC of UNIX

J-C. Soetens

1 Introducing UNIX

Unix was born in 1969 at Bell Laboratories (subsidiary of ATT) under the impetus ok Ken Thompson and Dennis Ritchie to meet a custom internal. Their objective was to develop a system interactive operating system for small machines equipped with possibilities comparable to those of large systems. Unix will then be written in C language (invented by Dennis Ritchie) From 1973, which will therefore make it portable on a large number of computers.

UNIX is a multi-tasking, multi-user operating system. As an operating system, its main role is to ensure an optimal distribution of resources (memory, processor (s), disks ...) between the different tasks of the different users.

The main functions of UNIX are:

- *Login and logout*: once the administrator of the system has registered a user, the system is in charge of checking the identity of the user when he wants to connect. A command interpreter (shell) allowing a dialogue user / system is then automatically activated.
- *Resource management* of the installation and sharing these resources between users and different tasks.
- *Data management*: this consists of the organization, maintenance and access to storage units (memory, hard disks, magnetic tapes ...)
- *Communication between users*: this is for example mail électronic or file transfers.
- *Programming environment*: these are the compilers (C, C ++, Fortran ...), text editors, programming assistance tools (debuggers ...).

1.1 Development of a working session

The *work session* is the time interval between the user login and logout. On connection, the system will ensure the identity of the user and if it is recognized, a dialog protocol (a shell) will automatically be established. The disconnection consists in indicating to the operating system that the session of work is finished.

1.2 Connection procedure

It is during this step that the user will have to identify himself from the system. This identification takes place in the way next :

- Display of the message **login:** after which you must enter your username (user id or uid).
- Display of the message **Password:** after which you must enter his password. This is not displayed while typing for prevent someone else from seeing it.

After connection, various messages from the administration of the system (words of the day, presence of mail in the box letters, etc ...) are displayed. The user is indeed ready to work when it receives the system prompt consisting of a marker at the beginning of the line (the prompt). This marker is variable depending on the machines (ex : **$** ou **nom_machine**[**numero**] > and can be redefining the user.

1.3 Password change

At the first connection or for safety, the user may have to change his password. This change is made with the command **passwd**. You can change your password by first entering the password current password then twice the new password. Example usage by user "gromit":

```
$ passwd
Changing password for "gromit"
gromit's Old password:
gromit's New password:
Enter the new password again:
```

If the new password is rejected, the reason may be that the second new password entry (for confirmation) was different from the first or, possibly, that the new password does not answer not to security requirements.

1.4 Logout procedure

This depends on the type of session that has been opened. If a graphical environment (such as the one created by X Windows, the multi-window environment) is in place, it usually exists a "logout" menu (or "exit") which allows you to exit this environment and thus to end the work session. Sometimes also this menu does not allow just quitting the graphical environment. Then you have to proceed to the disconnection step below. In the absence of a graphical environment, a simple command such as **logout** or **exit** is sufficient to end the working session.

2 Command interpreter

Once logged in, the user can submit commands to the system. *Any command entered will be interpreted by the interpreter of commands (or shell).* The term shell has been choosen for express the idea of an interface wrapping the kernel of the system and which establishes a communication between this kernel and the user.

There are many UNIX command interpreters. The main ones (found on most systems) are the Bourne-shell (sh), the C-shell (csh), the Korn Shell (ksh) and the T-CShell (tcsh). The choice of the interpreter activated on connection is made by the administrator of the system upon user registration and in most cases it this is the C-shell or the Korn-shell.

2.1 Command syntax

$\boxed{\textbf{name \_command} \text{ [} \texttt{options} \text{] [} \textit{arguments} \text{]}}$

- the separator character between the different elements of the command is blank (SPACE key on the keyboard).
- options usually start with the character - (sign minus) followed by one or more key letters. These options will modify the command behavior. Square brackets around arguments and options mean that these are optional.
- the arguments specify the objects (files or variables) on which command will apply.

While entering the command, the user can correct his typing using the DELETE or BACKSPACE keys. When the user has when the entry is finished, it submits the command to the system by pressing the ENTER key. Some shells have *mechanisms reminder of* commands which allow r úse (as is or after modification) a command previously used.

2.2 Elementary commands

ls	get the list of files in the current directory.
ls `-l`	`l` like long, gives all file attributes.
ls `-la`	`a` like all, also list files starting ç starting with the character "."
cat *f1*	display the contents of the file f1 on the screen.
cp *f1 f2*	copy from file *f1* to file f2 (**c** o **p** y).
mv *f1 f2*	rename the file f1 to f2 (**m** o **v** e).
rm *f1*	destroy file f1 (r e m ote).

Notes:

- All shells distinguish between lowercase letters and uppercase for commands and file names (unlike the from efunt MS-DOS).
- We have on-line help under UNIX allowing access to the rules of use and functionality of a command. For it, just issue the command:
  man *name _command.*

2.3 Special shell characters

When entering an order (**name _command** [`options`] [*arguments*]), all or part of the *argument* can be designated as c on subtle with the help of special characters:

?	denotes any any character.
$*$	denotes a sequence (which may be empty) of characters.
[...]	designates a character from the list.
[^...]	designates a character that does not belong to the list.

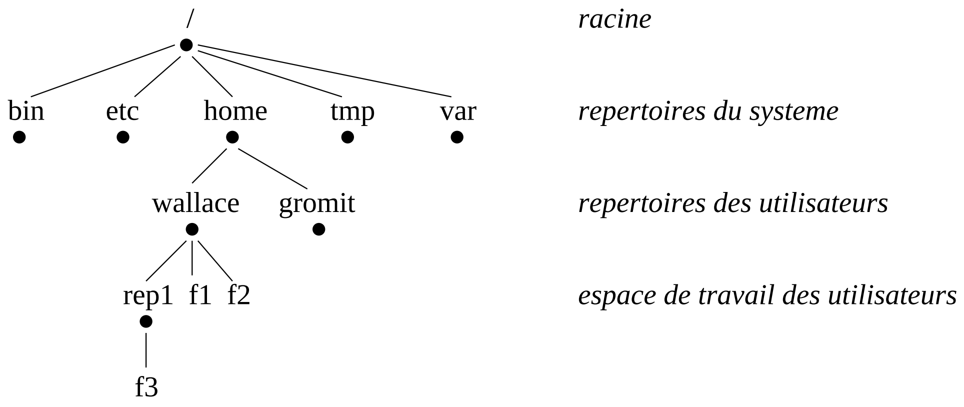
Example: either a current directory such as:
ls gives `file1.bin file1.txt file2.txt file10.txt file.txt readme zzz`

so

ls *file1* ∗	gives `file1.bin file1.txt file10.txt`
ls *file* ∗ *.txt*	gives `file1.txt file2.txt file10.txt file.txt`
ls *file* [*0-9*]∗ *.txt*	gives `file1.txt file2.txt file10.txt`
ls *???*	give zzz

3 Tree structure of UNIX files

The information unit managed by the system is the file. According to their use, the files are called directories or all files. short. A directory is a catalog of files containing their characteristics like access rights, size, date of creation... *The backbone of the system is a tree structure* of files (files) and directories (directories). Each user can create At will in his workspace of new files and directories.



3.1 Files and directories

Important Notions:

/ is the name of the tree root directory.

. designates the current directory.

.. designates the parent directory (just above).

Current directory: position in the tree at a given time.

Home directory: root of the tree structure of a user (home directory) and current directory each time connection. For example / home / gromit is the home directory of growled.

Absolute name: Uniquely denotes a file in starting from the root. Each file has a unique absolute name in the system. For example / home / wallace / rep1 / f3

Relative name: designates a file from the current directory. For example dir1 / f3 if the current directory is / home / wallace. Two files with the same name can coexist if they are in two different directories because they have an absolute name different.

3.2 Navigate in the tree structure

cd *name _directory*	allows to change directory (**c** hange **d** irectory).
cd ..	allows you to go back to the above directory.
cd	without argument, the command takes you to your home directory.
mkdir *rep1*	create the directory *rep1* (**m** a **k** e **dir** ectory).
rmdir *rep1*	delete the directory *rep1* (**r** e **m** ote **dir** ectory).

Notes:

- to create a file *f1* in a directory x is equivalent to (i) create the file *f1* and (ii) add the name *f1* in the list of names contained in x
- creating a directory y in a directory x is equivalent to (i) add the name y in the list of names contained in x and (ii) create a y file and associate an empty list with it.

3.3 Access rights to files

Each file (or directory) has a set of attributes defining the access rights to this file for all system users.

3.3.1 User classes

There are 3 classes of users who can optionally access To a file:

- the owner of the file (User).
- the group to which the owner belongs (Group).
- the others (Others).

When it is created, a file belongs to its author. The file owner can then distribute or restrict the access rights to this file as we will see later.

3.3.2 Types of access

For each user class, there are 3 types of access to a data file:

- in reading (Read).
- in Write.
- in execution (eXecute).

At the directory level, these rights mean:

- Read: right to list the files found in this directory.
- Write: right to create or delete a file located there.
- executed: right to traverse this directory.

3.3.3 Control and visualization of access rights

For this, we use the command **ls** `-l`.

```
home / wallace > ls -l
total 3
-rw-r - r-- 1 wallace usr 12 Dec 18 12:14 f1
-rw-r - r-- 1 wallace usr 297 Dec 18 12:14 f2
drwxr-xr-x 2 wallace usr 512 Dec 18 12:14 rep1
```

The first character specified if the file is a directory (character `d`) or a file (character `-`). The 9 characters following identify the access rights (presence of the right if letter `r, w` or `x`) in the order of user (u), group (g) and others (o).

3.3.4 Modification of access rights

Only the owner of a file can modify his access rights. For that, it uses the command **chmod** with the syntax next: **chmod** `who` ± `what` *name _file*
with `qui = u, g, o, a` and `what = r, w, x`, − to remove rights and + to add more.

Examples:

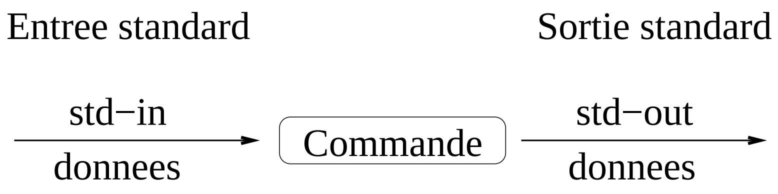
```
home / wallace > chmod or f1
home / wallace > chmod g + w f2
home / wallace > chmod go-rx rep1
```

The result of these commands is:

```
home / wallace > ls -l
total 3
-rw-rw ---- 1 wallace usr 12 Dec 18 12:14 f1
-rw-rw-r-- 1 wallace usr 297 Dec 18 12:14 f2
drwx ------ 2 wallace usr 512 Dec 18 12:14 rep1
```

4 Inputs / outputs and pipe

Usually the commands read the standard input and / or write to standard output. *By default, the standard input is the keyboard and the standard output is the screen.* It is possible to redirect this standard input and output to files using > and < characters.



4.1 Redirection of standard output

So that the result of a command is stored in a file at the instead of appearing on the screen, use the syntax:

$\boxed{\textbf{name \_command } [\ \texttt{options}\ ]\ [\ \textit{arguments}\ ] > \textit{file \_output}}$

If the file *file _output* does not exist, it is created and will contain the order result. If it existed before, its content is is overwritten. If we want this content is preserved and add the result of a command, you must use the redirection >>

4.2 Redirection of standard input

Likewise, instead of providing data by entering it on the keyboard, these data can be read from a file with the syntax:

$\boxed{\textbf{name \_command } [\ \texttt{options}\ ]\ [\ \textit{arguments}\ ] < \textit{file \_output}}$

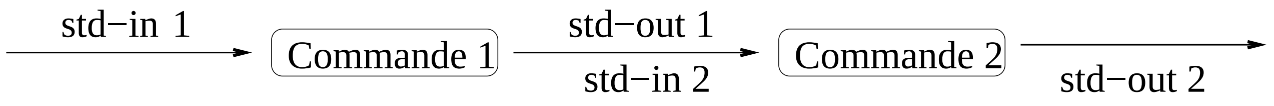
Example:
wc `-l` *file1*: indicates on the screen (standard output) the number of lines of the file file1.
ls `-l` *file1* > temp; **wc** `-l` *temp*: same result than the previous command.

4.3 Pipe

In the same order of ideas as the redirection of standard / output, we can use the *mechanism of pipe* (tube) which allows to take the standard output of a first command and redirect it to the standard input of a second order. The syntax uses the character | as follows:

$\boxed{\textbf{name \_command1 } [\ \texttt{options}\ ]\ [\ \textit{arguments}\ ]\ |\ \textbf{name \_command2 } [\ \texttt{options}\ ]\ [\ \textit{arguments}\ ]}$

This sequence of commands can be schematized as follows:



Example:
ls `-l` *file* ∗ | **wc** `-l`

5 Search tools

Unix offers powerful file search or search tools in files.

5.1 Search for strings in a file

The **grep** command displays the lines of the data files in arguments which contain a given pattern:

grep `options` pattern *files*

Example:
grep subroutine ** .f*
allows to display on the screen all the lines of the files of the current directory with extension ".f" and containing the pattern "subroutine".

Basic options:

`-i`	ignores the lowercase / uppercase distinction.
`-v`	display lines which do not contain the pattern.
`-n`	print out lines and their numbers that do not contain the pattern.

5.2 Search for a file

The **find** command descends recursively in subtrees of directories, seeking to apply to files specified by one or more selection criteria (name, type, date of modification, etc.), a given command. **find** is used as ç on:

find *list _of _r éxpression directories*

where *list _of _ directories* is the list of tree roots To browse and *expression* is a series of options expressing the file selection criteria and actions to apply to them. When the criterion is true, the action is executed.

Example of selection criteria:

`-name "pattern"`	true if a current filename contains the pattern string.
`-user username`	true if a current file belongs to user username.
`-mtime n`	true if the file has been modified in the last n days (+ n to express "n and more" and -n to express "n and less").

Example of actions to be performed on the selected files:

`-print`	display the name of the current file.
`-ls`	list of information about the current file.

Examples:
find. -name "* .f" -print
allow to search recursively from the current directory "." all files with the ".f" extension.

find / home / gromit -name "*" -print | xargs -i grep -i "Wrong Trousers"
allows recursive search from directory / home / gromit all files and display the lines that contain the string of "Wrong Trousers" characters without distinguishing between upper and lower case.

Annex B

START PROGRAMMING IN FORTRAN

J-C. Soetens

University of Bordeaux

First simple program in Fortran

Name of the file : ex1.f

```
123456789012345…                                                 72
      program example_1
c     my first program
      write(*,*) ''Hello world''
      end
```

One instruction per line.
Upper and lower case are not significant, blank lines and spaces are not significant.

To create an executable, you need to compile the code.
If the compiler is gfortran :

your prompt > gfortran –o ex1 ex1 .f

To execute the program, submit the new command to the system :
your prompt > ./ex1

Fortran operators

**	raise to the power of
*	multiplication
/	division
+	addition
-	subtraction

Types of variables

It is recommended to impose the declaration of all variables : *implicit none*.
Types : integer, double precision, logical, character.

Name of the file : ex2.f

```
      program example_2
      implicit none

      double precision TC, TK, Kfactor
      parameter(Kfactor = 273.15d0)

c     input
      write(*,*) ''Enter the temperature in C :''
      read(*,*) TC
      if (TC .lt. (-Kfactor)) then
        write(*,*) ''this temperature do not exist''
        STOP
      end if
c     algorithm
      TK = TC + Kfactor
c     output
      write(*,*) ''The temperature in Kelvin is : '', TK
c     output using a format
      write(*,''(The Temperature in Kelvin is :'', F8.2)') TK

      end
```

Intrinsic functions

Some functions are so important that they are provided as part of the language.
As we will systematically use real variables in double precision, the needed intrinsic functions name will start with the letter ‘D’.

Examples :

`DABS(X)`	absolute value of any X
`DCOS(X)`	cosine of argument in radians
`DSIN(X)`	sine of argument in radians
`DTAN(X)`	tangent of argument in radians
`DACOS(X)`	inverse cosine in the range (0,ð) in radians
`DASIN(X)`	inverse sine in the range (-ð/2,ð/2) in radians
`DATAN(X)`	inverse tangent in the range (-ð/2,ð/2) in radians
`DEXP(X)`	exponential function
`DLOG(X)`	natural logarithm: if W is real it must be positive,
`DLOG10(X)`	logarithm to base 10
`DSQRT(X)`	square root function

Logical controls

The `if` statement is the way of changing what happens in a program according to a condition.

Syntax :

```
      if (logical expression)then
c       instructions(s) in case the logical expression is true
        …
      else
c       instruction(s) in case the logical expression is false
        …
     end if
```

Main operators :

`.lt.`	less than
`.le.`	less than or equal
`.eq.`	equal
`.ge.`	greater than or equal
`.gt.`	greater than
`.ne.`	not equal
`.not.`	not
`.and.`	and
`.or.`	inclusive or

Loops

In case you need to repeat a set of instructions, different ways exist to do this repetition.

Case 1, you know the number of repetitions : `do loop`.

```
     do variable  = start, stop [,step]
c      instructions to do
       …
     end do
```

with :

variable	is an integer variable
start	is the initial value var is given
stop	is the final value
step	optional, is the increment by which var is changed

Case 2, you do not know the number of repetitions : `do while`

```
      do while (logical expression)
c       instructions
        …
      end do
```

WARNING ! Possibly an infinite loop if the logical expression is always true (never false).

Arrays

Important in scientific programming: manipulation of vectors and matrices.

Examples of declarations :

```
      integer V(10)
      integer M(10,10)
      double precision N(2,50)
```

V is a vector of 10 (integer) elements.
V(1) is a pointer to the first element, V(10) is a pointer to the tenth element.

M is a table of 10 x 10=100 (integer) elements.
In the memory of the computer, it is a stack of elements with the first argument running in first, so the order is M(1,1), M(2,1)…M(10,1), M(1,2), M(2,2)…M(10,2) …M(8,10), M(9,10), M(10,10).

N is a table of 2 x 50=100 (double precision) elements.

Name of the file : ex3.f

```
      program example_3
      implicit none

      double precision M(10,10)
      integer i, j

c     initialize all the elements of the table M to zero
      do j=1, 10
        do i=1, 10

          M(i,j) = 0.0d0

        end do
      end do

c     use of table M to to something…

      end
```

Functions

As for intrinsic functions, you can define our own functions for use in a program. This is a very powerful feature because it allows to writte code once while it can be used many times. Such own functions can be programmed and tested separately, even build a library.

Name of the file : ex4.f

```
      program example_4
      implicit none

      double precision V1(3), V2(3)
      double precision scalar_product
      integer i
```

```
c     initialize the 3-D vectors V1 and V2
      do i=1, 3
        read(*,*) V1(i), V2(i)
      end do

      write(*,*) scalar_product(3, V1, V2)

      end
```

Name of the file : sp.f

```
      function scalar_product(n, A, B)
      implicit none

       integer N
       double precision A(N), B(N)
      double precision scalar_product
      integer i

      scalar_product = 0.0d0
      do i=1, N
        scalar_product = scalar_product + A(i)*B(i)
      end do

      end
```

To test the function scalar_product solely (so only the Fortran syntax) :

your prompt > gfortran –c sp.f

To build an executable with the two files :

your prompt > gfortran –o ex4 ex4.f sp.f

Subroutines

Subroutines are very similar to functions but they do not return a value.

Example of code of a subroutine :

```
c      subroutine MY_FIRST_SUBROUTINE(argI, argJ,…,argZ)
       implicit none

c      declarations of arguments
       …

c      work to do
       …

       end
```

Somewhere in a program

```
c      programm example_5
       implicit none

c      declarations
       …

c      core of the program
       …
       call MY_FIRST_SUBROUTINE(arg1, arg2,…,argN)
       …

       end
```

Many more things to explain !
List of arguments, declarations, local and global variables, etc.

If you want to know (much !) more :

http://www.idris.fr/formations/fortran/

Training exercises :

Write, compile and test your own programs :

1) Write a program that accepts two vectors of dimension n, then calculates and displays the sum of the two vectors.

2) Same as problem 2) but your program must read the vectors components from a file.

3) Write a program that accepts two vectors of dimension n, then calculates and displays their scalar product.

4) Write a program that accepts the coefficients of a quadratic equation (ax2+bx+c=0) and finds the roots. Consider all possible cases including the complex one.

5) Re-write the sum and scalar product of two vectors as ‘subprograms’.

6) Write a program to calculate the integral of a function using the ‘trapeze ‘ approximation.

7) Write a program to sort a list of values in ascending order.

8) etc.