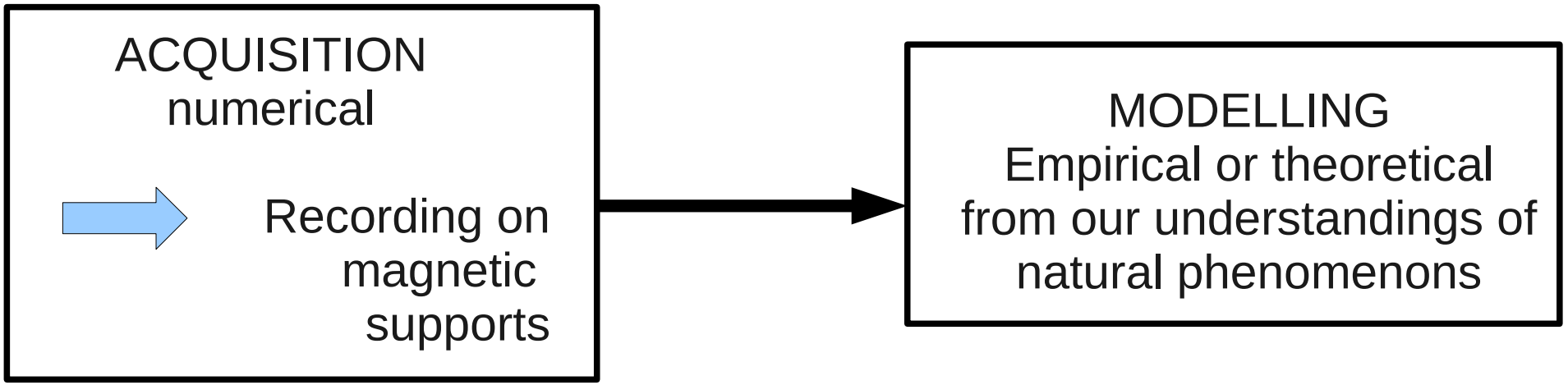
FORTRAN programming

General introduction

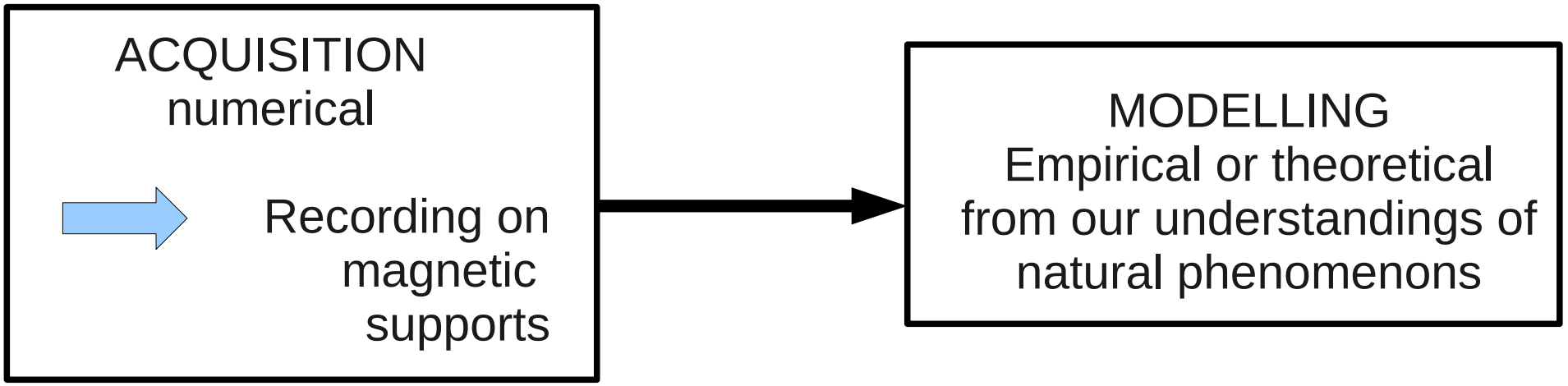
Virginie DURAND and Jean VIRIEUX

Why learning programming ???

Why learning programming ???



Why learning programming ???



- *IMPORTANT to master the processing line*

  acquisition → *modelling*

- *To know how to modify the programs at your disposal to do the job asked.*

Computer environment

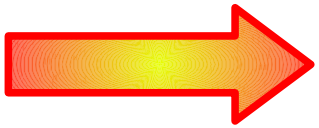
- ***CPU*** : processor controlling the whole system
- ***Main memory*** : enable the temporary memorization of the data during program execution
- ***External mass storage*** : storage of informations on the long range (hard disk, CD-ROM,...)

Integrated tools

- Excel, Matlab, Scilab, Octave,...
- Allow easy and quick data handling
- Enable quick test of an idea
- **BUT**
  - They are slow
  - They can have some difficulties during heavy applications

Because each step is analysed by the computer each time it is met !!

Solution : avoid this repeated analysis !!! (compiled tools)



Cycle Steps/Compilation+Link editing /Execution

Languages

- Why using a programming language ??

Languages

- Why using a programming language ??

Language = practical way to give instructions to a computer

- Language setting
  - Key words
  - Handleable tools
  - Syntax rules
  - Logical structures

Programming is writing a text observing language rules, and likely to solve a given problem

Languages

- Sequential language (declarative programming)
  - Program = series of instructions brought together in blocks
  - There are some conditional jumps ==> back to an instruction block if the condition is true
  - C, ADA, FORTRAN

Languages

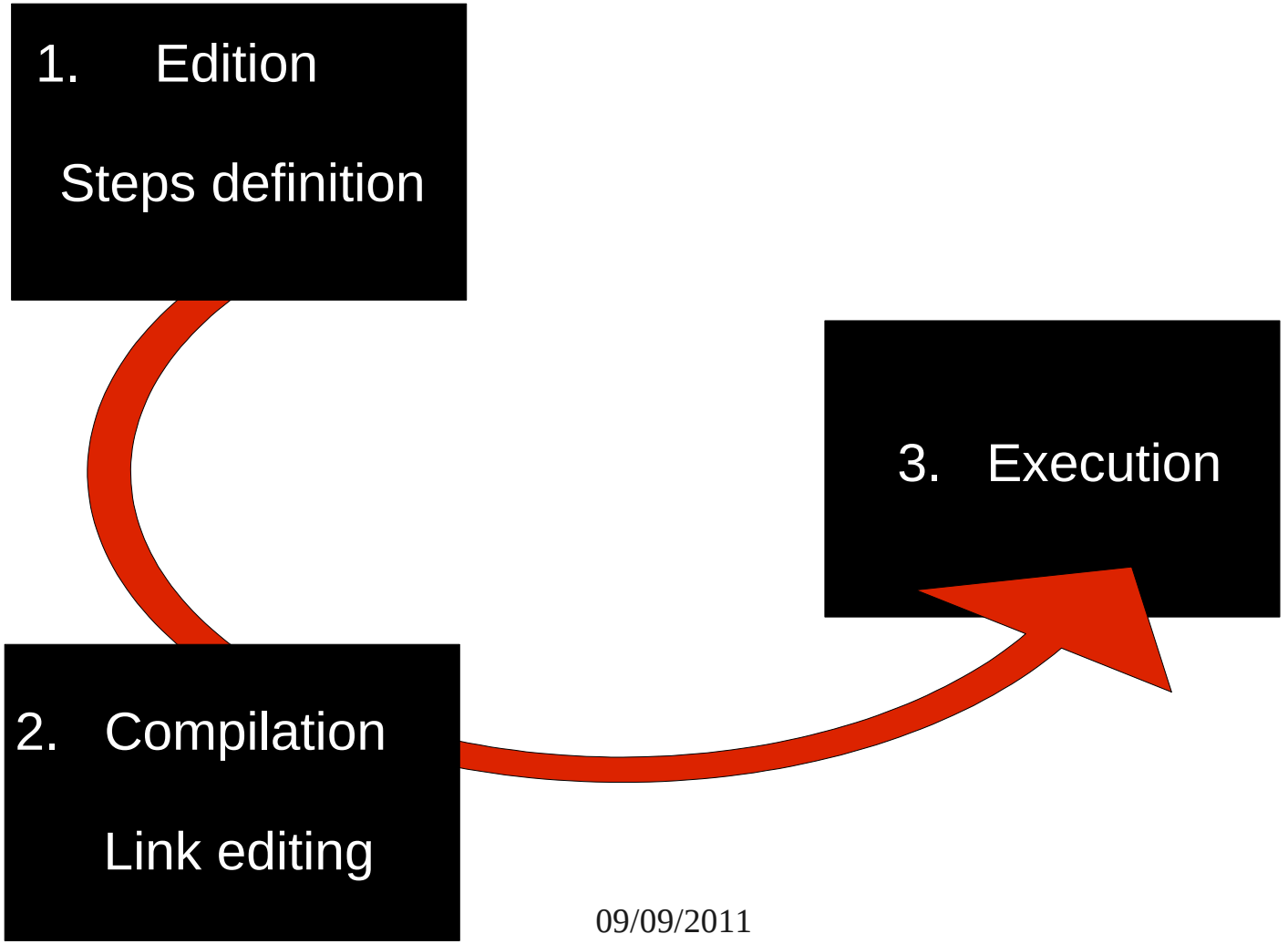
- Object-oriented language
  - Program built with relations
  - Relations : define objects and links between objects

    ==> user defines properties, the language makes the deductions
  - C++, ADA, smalltalk

Why fortran ?

- Archetypal scientific language since 1957
- Portability on various architectures
- Allows quick executions

Program edition, Compilation et link editing, Execution



Program edition, Compilation et link editing, Execution

- ***Compilation*** : translation of the program in binary

- ***Link editing*** : process allowing to create runnable (executable) files from the objects files *(intermediate files)*

  Link editor links object files with the environment

1. Edition

toto.f90

```
program toto
do i = 1,100
write(*,*) i
enddo
stop
end
```

2. Compilation

gfortran -c toto.f90

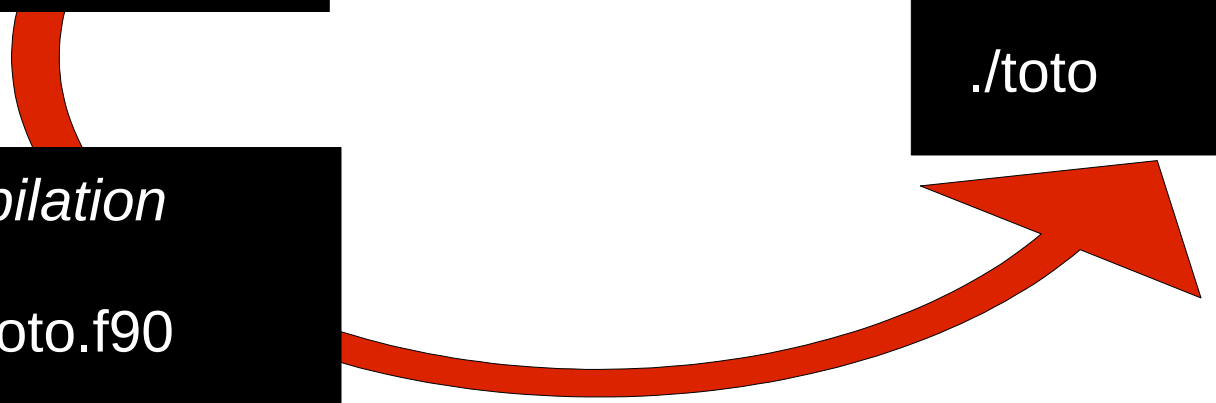
==> toto.o

Link editing

gfortran -o toto toto.o

3. Execution

./toto



Basic Unix (*shell*)

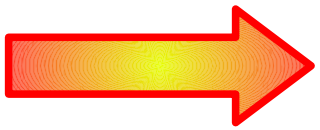
Shell definition and aim

What ?

- Interface for the system user
- Command interpreter

Why ?

- Interactive use : command line
- Programming (script conception)
- **File handling**



Automate tasks

Various types of shell (cf P.Fuchs)

- Bourne shell(/bin/sh) :
  - Shell standard
  - The most compact and the easiest
  - It is on all systems
- Bourne again shell (/bin/bash)
  - Extensive version of Bourne shell
- C shell (/bin/csh)

How to travel in the directories

- ***pwd*** : displays the current directory path from your home
- Absolute or relative path :

  ex : If I am in toto and I want to go in tata, 2 possibilities :

  - cd ../tata = relative path
  - cd ~/tata = absolute path (+++ if we have to go back up a lot, be carefull if you move your file)

Permission modifications

- ***Chmod -u permission file***

- u = user
- Permission :

r	read	4	+/-
w	write	2	+/-
x	run	1	+/-

chmod +r+w+x file <==> chmod 777 file

Shell in command line interface

- 2 states:
  - Work
  - Inactivity : - waits for orders, executes them, waits for a new order
    - display a prompt :

Interpretation and execution of the commands

- Ex : ls -l file
- To interpret a line, Bash splits it up in words :
  - 1st word = command name : *ls*
  - Then the arguments of the command = data processed by the command : *file*
  - Options will change the behaviour of the command : *-name_option : -l*
  - End of the command :
    - « **;** » if several commands on the same line

    or
    - Next line

How does shell find the commands ??

- Example : I write « Hello »

  shell will look at ≠ places :

(1) Is « Hello » an integrated command ?

  - If yes : execution
  - If no --> (2)

(2) Reads the content of a variable, PATH (points out the path of the command)

In PATH : /usr/bin → look for: /usr/bin/Hello

/bin → /bin/Hello

(3) If (2) negative ⇨ error message : *Hello:command not found*

Arguments and options

- ***Argument*** = series of characters given to a command ---> tells how to behave
- ***Options*** = special case of arguments ---> all kind of informations
- ***Metacharacter*** = character with an other signification than its literal one.

  ex : replace other characters :

  - * = any character/group of characters
  - **?** = 1 character
- « **** » : - prevent the special interpretation of a character
  (* --> écrit *)

  - at the end of a line ==> command continues at the next line

Command hystory

- Shell keeps an history of the commands ===> history number

  command *history*

- arrows : back up/go down in the history
- Recall of a command :
  - ***!x*** : run again the command N° x in the history
  - ***!!*** : run again the last command
  - ***!cp*** : run again the last command beginning with cp

Process

- Run a programm is create a process
- Process = execution of a series (more or less long) of instructions (program, script,...)
- Unix commands listing the processes : *pstree, ps, ps -l, ps aux*

  Ex : *ps* gives:

```
[durandv@lgit-1197]$  ps
 PID  TTY          TIME     CMD
3195 pts/1    00:00:00    bash
3366 pts/1    00:00:00    okular
3375 pts/1    00:00:00    ps
```

State of a process (cf P.Fuchs)

- Foreground process (fg) :
  - shell waits for the death of the son process to take the hand again
  - In abstentia, each command is run fg
- Background process (bg) :
  - We keep the hand in the shell
  - To run in bg : *name_commande &*
  - Usefull to run a long process

- *Rq :* if we want to be able to close the shall during a pgm is running : ***nohup*** *program*

!!! put the errors in a file !!!

Commands acting on a process state (cf P.Fuchs)

- ***jobs*** : lists the jobs of a shell and their state
  - Option *-l* gives the PID
  - Allocates a n° *x* of job, that you can recall with *%x*
- ***Kill*** : kills a process
  - *kill %1* : kills the process n°1
  - *kill -9 PID* : kills the process
- ***Ctrl-Z*** : stops the job
- ***Ctrl-C*** : kills the job

top command

owner

PID

Ressources

Process name

durandv@lgit-1197:~/Documents/these

Fichier Édition Affichage Rechercher Terminal Aide

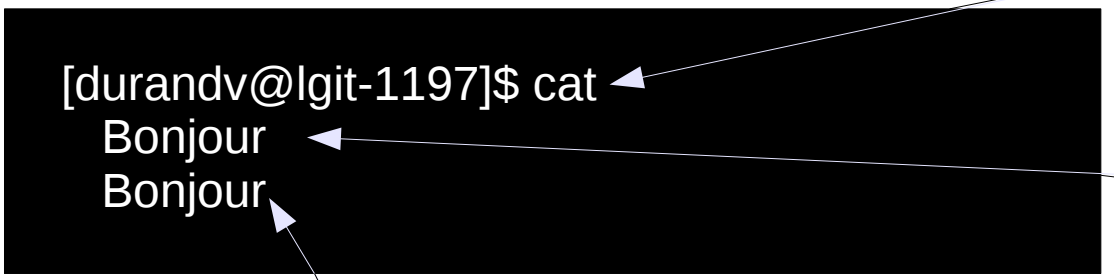
```
Tasks: 224 total,   1 running, 223 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.4%us,  0.7%sy,  0.0%ni, 97.9%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3539340k total,  1278452k used,  2260888k free,   106420k buffers
Swap:  4094972k total,        0k used,  4094972k free,   691952k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2891 durandv   20   0  496m 138m  27m S  5.3  4.0   3:56.76 firefox
 2191 root      20   0  129m  31m  13m S  1.3  0.9   2:10.88 Xorg
 3799 durandv   20   0 86784  15m  10m S  1.0  0.4   0:02.14 npviewer.bin
 2596 durandv   20   0  249m 105m  61m S  0.7  3.1   1:26.68 soffice.bin
 2911 durandv   20   0  421m  54m  25m S  0.3  1.6   0:09.50 thunderbird-bin
 3174 durandv   20   0  130m  13m  10m S  0.3  0.4   0:03.99 gnome-terminal
 3859 durandv   20   0  2728 1068  764 R  0.3  0.0   0:00.05 top
    1 root      20   0  2876 1364 1152 S  0.0  0.0   0:01.96 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:00.21 ksoftirqd/0
    4 root      RT   0     0    0    0 S  0.0  0.0   0:00.45 migration/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:00.29 migration/1
    7 root      20   0     0    0    0 S  0.0  0.0   0:00.04 ksoftirqd/1
    8 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/1
    9 root      RT   0     0    0    0 S  0.0  0.0   0:00.29 migration/2
   10 root      20   0     0    0    0 S  0.0  0.0   0:00.12 ksoftirqd/2
   11 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/2
   12 root      RT   0     0    0    0 S  0.0  0.0   0:00.71 migration/3
   13 root      20   0     0    0    0 S  0.0  0.0   0:00.03 ksoftirqd/3
   14 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/3
   15 root      20   0     0    0    0 S  0.0  0.0   0:07.24 events/0
   16 root      20   0     0    0    0 S  0.0  0.0   0:00.07 events/1
   17 root      20   0     0    0    0 S  0.0  0.0   0:00.04 events/2
   18 root      20   0     0    0    0 S  0.0  0.0   0:00.02 events/3
   19 root      20   0     0    0    0 S  0.0  0.0   0:00.00 cpuset
   20 root      20   0     0    0    0 S  0.0  0.0   0:00.01 khelper
   21 root      20   0     0    0    0 S  0.0  0.0   0:00.00 netns
   22 root      20   0     0    0    0 S  0.0  0.0   0:00.00 async/mgr
   23 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pm
   24 root      20   0     0    0    0 S  0.0  0.0   0:00.00 sync_supers
   25 root      20   0     0    0    0 S  0.0  0.0   0:00.00 bdi-default
   26 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kintegrityd/0
   27 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kintegrityd/1
[durandv@lgit-1197 these]$
```

[Yahoo! Mail (vi_mari... | [Courrier entrant - M... | cours_program | cours1_fortran_pres... | durandv@lgit-1197:... | durandv@lgit-1197:... | [stress1kmIzmit.jpg ...

Flux

- Flux = flow of data coming in and out of the processes (or programs) :
  - Standard input (*stdin*) : keyboard
  - Standard output (*stdout*) : screen
  - Error output (*stderr*) : Screen



Commande cat without argument

Bonjour is written with the keyboard (stdin)

Bonjour is displayed on the screen (stdout)

Flux redirection

- ***A > file*** : stdout of A is put in *file* overwritting its content or creating it
- ***A >> file*** : stdout of A is put at the end of *file*
- ***A 2 > file*** : stderr is put in *file* (overwritting, creation)
- ***A < file*** : executes A with the content of *file* in stdin
- ***A | B*** : executes A then sends stdout(A) in stdin(B)

  Ex : *ls -l* | *sort* : gives the list of the files sorted

Some filter commands

- ***head file*** : displays the 1st lines of *file*
- ***tail file*** : displays the last lines of *file*
- ***grep expression file*** : displays all the lines of *file* with *expression*
- ***sort file*** : sorts each line of *file*
- ***wc -l fichier*** : displays the nbr of lines of *file* (option ***-c*** : nbr of characters, ***-w*** : nbr of words)
- ***cat file1 file2 ...*** : concatenation of *file1*, *file2*,...

Variables

- Access to the content of a variable : ***$var***

  Ex : *echo $HOME* --> display the content of the variable HOME

- Some environment variables

  (contain the user environment caractéristics)

  - ***PATH*** : stores the access path to find the command asked by the user
  - ***USER*** : stores the name of the user
  - ***HOSTNAME*** : stores the name of the machine
  - ...

Shell programming

- **Script** = text file, serries of commands
- Execution of the script : each command is analyzed and translated in machine language
- *!!! Comment your program !!!*
- Make the script runnable !!

  chmod +x file.sh
- Write where is the interpreter on the 1st line :

  #!/bin/bash
- In an other line # = commentaire

Summary

- Main commands to travel in the directories

Summary

- Main commands to travel in the directories (cd, mkdir, cp, rm,...)

Summary

- Main commands to travel in the directories (cd, mkdir, cp, rm,...)
- How to change the permission of a file

Summary

- Main commands to travel in the directories (cd, mkdir, cp, rm,...)
- How to change the permission of a file
- Never forget to organize your work space !

Summary

- Main commands to travel in the directories (cd, mkdir, cp, rm,...)
- How to change the permission of a file
- Never forget to organize your work space !
- Remember to open your 3 windows when programming in fortran

 3 steps :

Summary

- Main commands to travel in the directories (cd, mkdir, cp, rm,...)
- How to change the permission of a file
- Never forget to organize your work space !
- Remember to open your 3 windows when programming in fortran

3 steps :

- **Writing the program**
- **Compiling and link editing**

  gfortran -c toto.f90, gfortran -o toto toto.o

- **Execution**