Introduction to HPC Cluster (day 1)

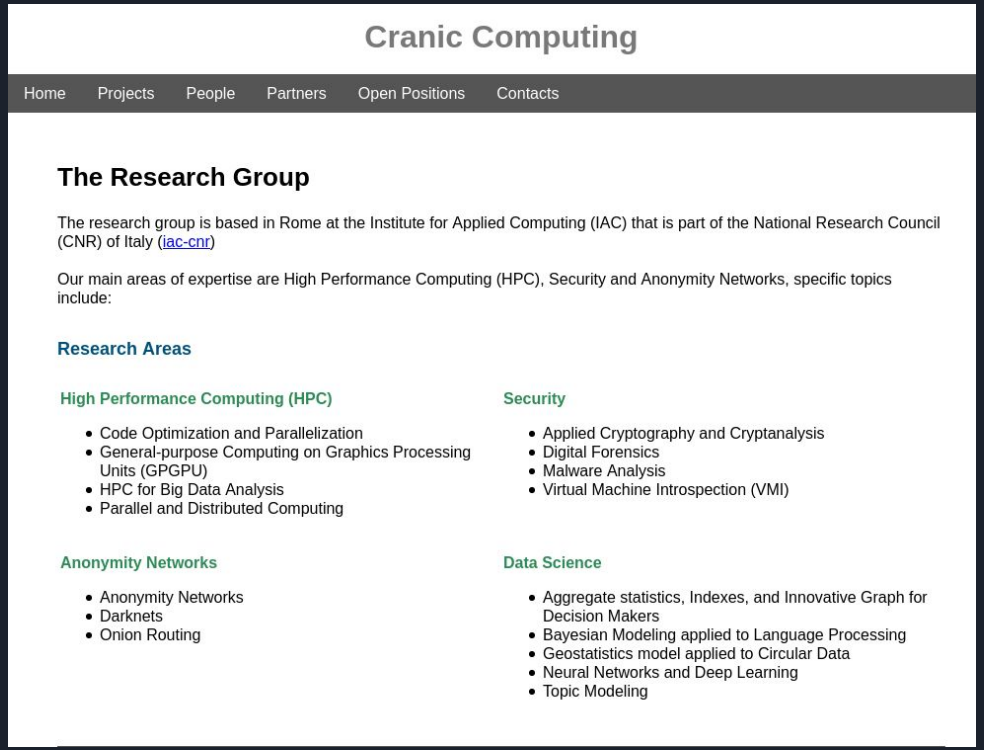
First Summer School in HPC and AI
7th - 8th June 2021 Free University of Bozen

Marco Cianfriglia <m.cianfriglia@iac.cnr.it>

Who I am


Istituto per le Applicazioni del Calcolo "Mauro Picone"
NVIDIA
CUDA
DE CIFRIS
Cranic Research Group
www.cranic.it
ROMA TRE
UNIVERSITÀ DEGLI STUDI
SAPIENZA
UNIVERSITÀ DI ROMA

www.cranic.it



Cranic Computing

Home Projects People Partners Open Positions Contacts

The Research Group

The research group is based in Rome at the Institute for Applied Computing (IAC) that is part of the National Research Council (CNR) of Italy (iac-cnr)

Our main areas of expertise are High Performance Computing (HPC), Security and Anonymity Networks, specific topics include:

Research Areas

High Performance Computing (HPC)

- Code Optimization and Parallelization
- General-purpose Computing on Graphics Processing Units (GPGPU)
- HPC for Big Data Analysis
- Parallel and Distributed Computing

Security

- Applied Cryptography and Cryptanalysis
- Digital Forensics
- Malware Analysis
- Virtual Machine Introspection (VMI)

Anonymity Networks

- Anonymity Networks
- Darknets
- Onion Routing

Data Science

- Aggregate statistics, Indexes, and Innovative Graph for Decision Makers
- Bayesian Modeling applied to Language Processing
- Geostatistics model applied to Circular Data
- Neural Networks and Deep Learning
- Topic Modeling

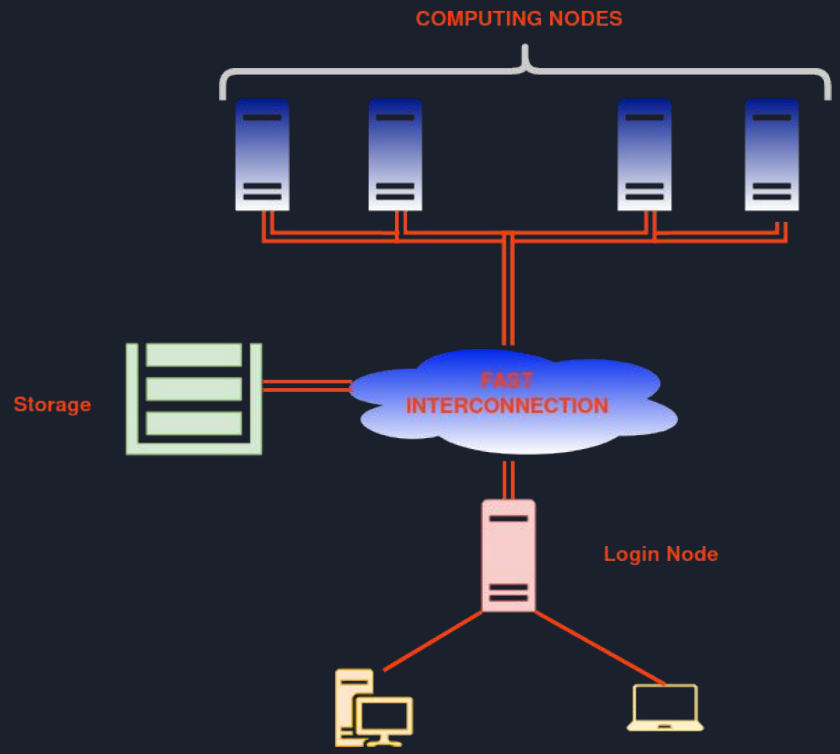
Agenda

- A brief introduction to HPC
- Access to Unibz cluster
- Unix/Linux fundamentals
- Bash scripting

A brief introduction to HPC

What is an HPC Cluster

- A collection of computer, called ***nodes*** connected by a **fast** network
- There may exist different types of nodes for **dedicated** tasks (example, equipped with GPUs)
- A special node, called ***login-node*** where users login and submit jobs
- A storage area **shared** among nodes



When do we need an HPC cluster?

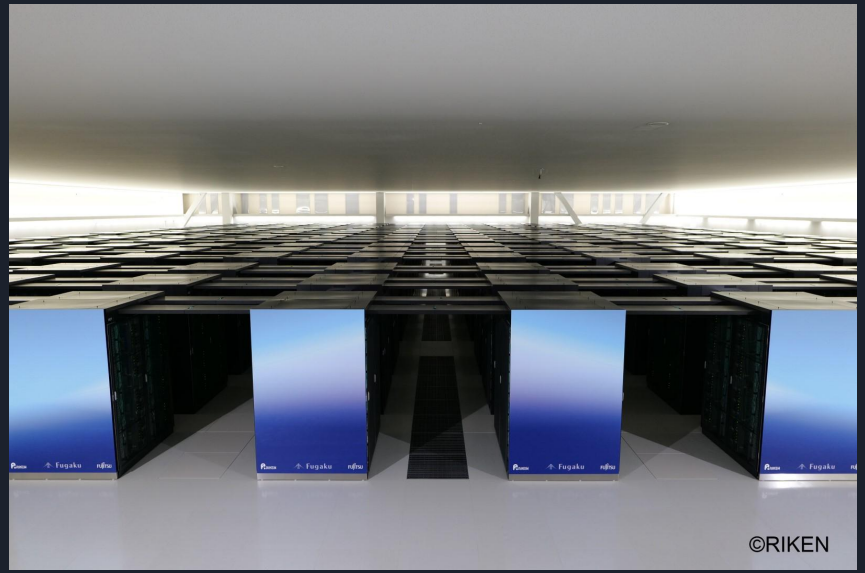
- Computations require much more **memory** than the amount available on your computer
- Simulations that need to be executed **many times** with different inputs
- Programs that may be run in **parallel** (MPI, OpenMP) speeding up the computation
- Some applications may benefit of hardware accelerators, like **GPUs**

TOP 3 HPC in the world (November 2020 list)

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438

https://www.top500.org/lists/top500/list/2020/11/

HPC in numbers



Soruce blog.global.fujitsu.com

Fugaku Supercomputer - 1st HPC system in the world - Top500 November 2020

158976 nodes
7630848 cores
~ 5PB of Memory (5087232 GB)
~442010 TFlop/s

“The six-year budget for the system and related technology development totaled about $1 billion”
(The Ney York Times 2020-06-22)

HPC cluster vs laptop

12	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100, Cray/HPE Swiss National Supercomputing Centre (CSCS) Switzerland	387,872	21,230.0	27,154.3	2,384

In 2017, Piz Daint was in third position of the Top500 list:

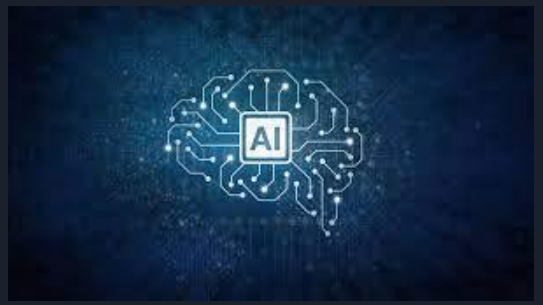
“what Piz Daint can process in one day, a laptop would take 900 years.” (*)

(*) https://www.swissinfo.ch/eng/speed-processing_swiss-supercomputer-third-fastest-in-the-world/43271536

HPC Applications

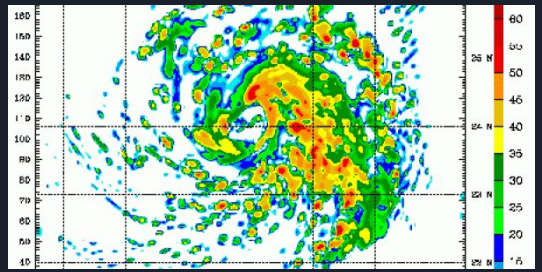
Source Forbes

Source Forbes

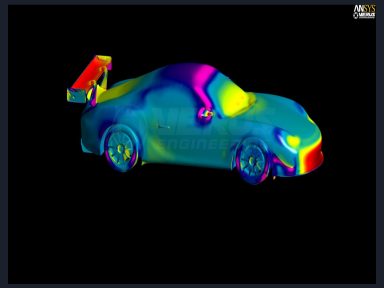


Source Wikipedia

Source Ohio.edu



Source Colorado.edu



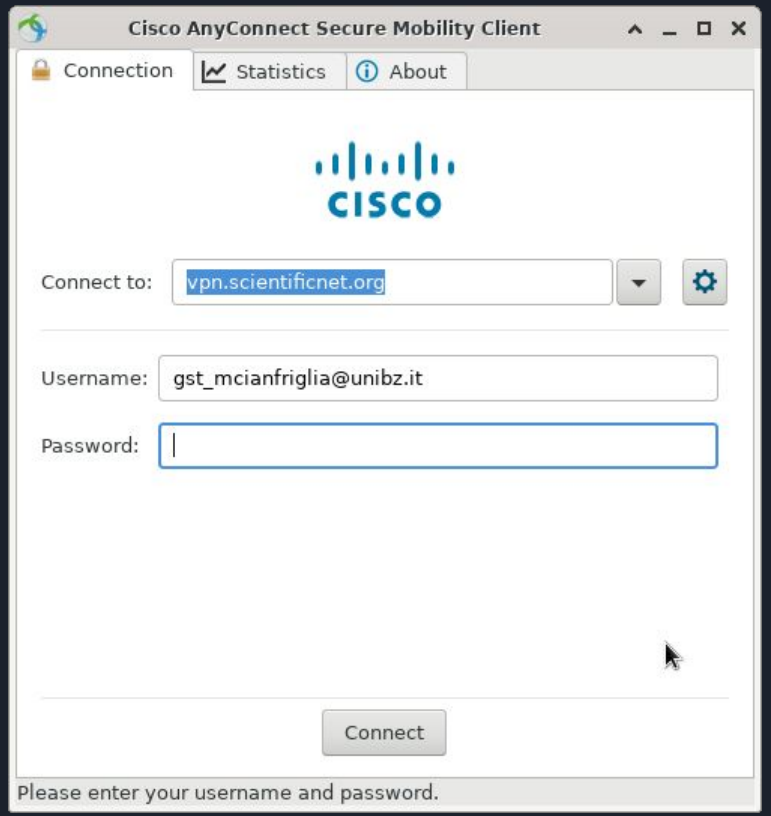
Source Wikipedia

Access to Unibz Cluster

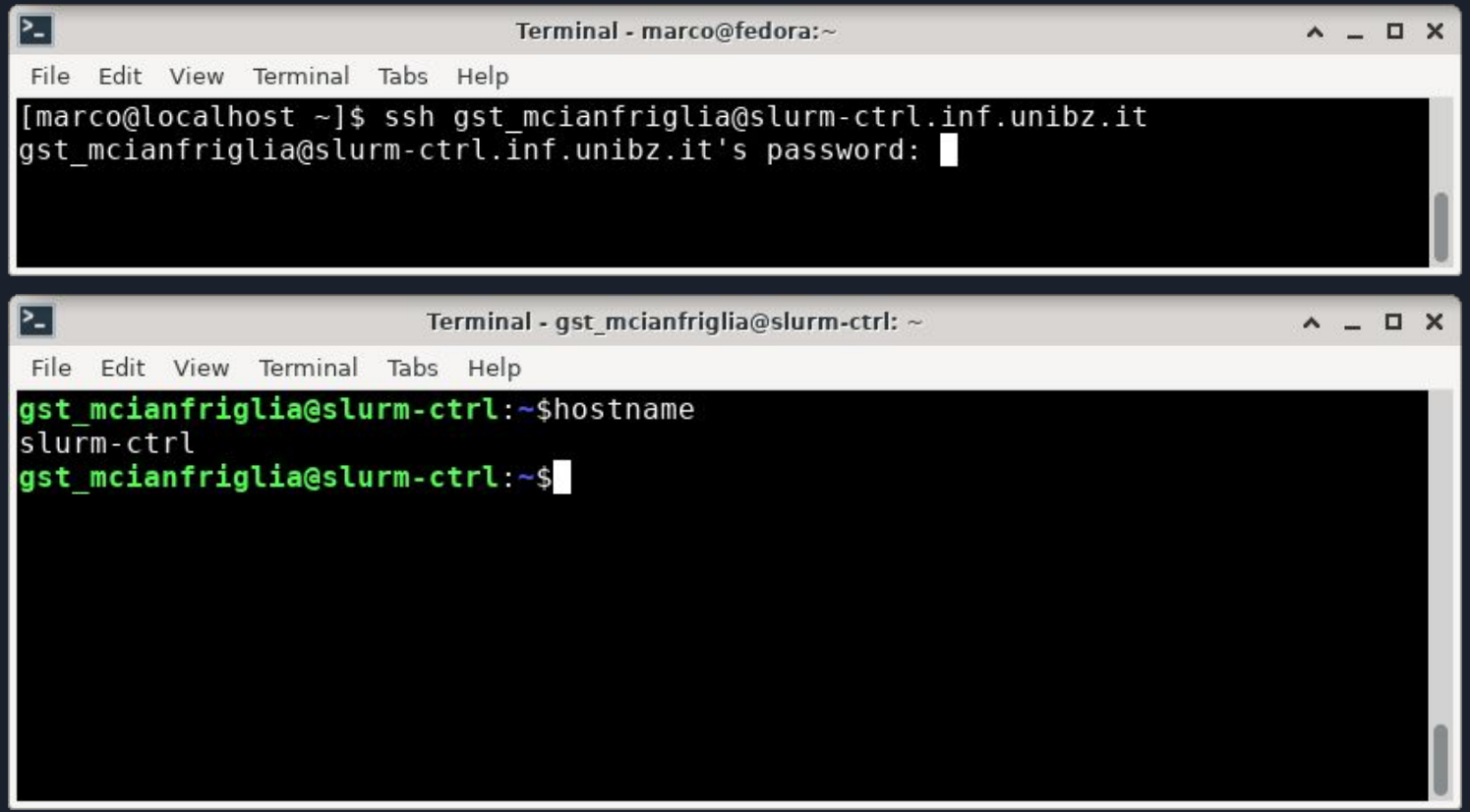
Connect to login node: slurm-ctrl.inf.unibz.it

1. Connect to VPN with your credentials (if needed).
   Please use your full username (@domain)

2. Connect to the login node:
   ssh <username>@slurm-ctrl.inf.unibz.it



Login node


Terminal - marco@fedora:~
File Edit View Terminal Tabs Help
[marco@localhost ~]$ ssh gst_mcianfriglia@slurm-ctrl.inf.unibz.it
gst_mcianfriglia@slurm-ctrl.inf.unibz.it's password:
Terminal - gst_mcianfriglia@slurm-ctrl: ~
File Edit View Terminal Tabs Help
gst_mcianfriglia@slurm-ctrl:~$hostname
slurm-ctrl
gst_mcianfriglia@slurm-ctrl:~$

Unibz Storage

HOME

- It is **shared** among all the nodes

SCRATCH

- it is available on */scratch*
- It is **local** to each node
- It is a **temporary** storage

For the hands-on we will use the home partition, unless specified.

Scratch partition

It is a temporary fast, unreliable storage, that can be used in several HPC scenarios:

- You need to store some **temporary** data during a long computation
- Your input dataset is **too big** to be stored on a permanent storage
- Your application uses **intensively I/O** and you may take advantage of **fast storage**

Two scratch deployment options :

- Local to the node
- Shared among cluster nodes

Unix/Linux fundamentals

History

- Origins and history
  - Developed in 70 by Bell Labs
  - Extreme portability and flexibility, it is written in C
- Many proprietary distributions/implementations
  - AIX (IBM), SOLARIS (SUN),
  - Mac OS X (Apple), HP-UX (HP)
- Many Open-source distributions/implementations
  - Linux (Debian, Suse, Red Hat, Ubuntu, …)
  - FreeBSD, OpenBSD, …
  - Android

Features

- **multi-tasking**
  - Many programs can be executed concurrently
  - The computational and memory resources are shared among processes
- **multi-user**
  - Different users can work concurrently on the same machine
- **memory protection**
  - Every process can have exclusive access to an area in memory. This mechanism improves security and stability of the whole system

Users

- On Unix every user can start working only after a successful authentication. This usually requires a username and a corresponding password
- To each user are assigned some resources (e.g. disk space)
- An unique identifier **UID** is assigned to each user together with the Group Identifier **GID**. These IDs are used by the system to manage the permissions
- There exist some users with special privileges. The default is **root**
  - it has maximum privileges
  - It can do (almost) everything on the system
  - It is characterized by UID=0

Shell

- They provide an interface between the operating system and the users.
- They are special programs that allow users to run and manage other programs (for example: redirect input/output, scripts, etc)
- There exist several implementations **sh, csh, ksh, tcsh, bash** but the latter two are the most used

Secure SHell (SSH)

- SSH is a program for **logging** into a remote machine and for **executing** commands on it
- It establishes a **secure** encrypted channel between two hosts over an **insecure** network
- Examples:
  a. ***ssh <username>@host*** # login on host with user <username>
  b. ***ssh <username>@host mkdir -p testdir*** # launch mkdir command on host
  c. ***ssh -l username host*** #Same as a)
  d. ***ssh -i <path-to-private-key> username@host*** # You may use public key authentication
  e. ***ssh -L8888:localhost:8890 username@host*** # Some advanced features - port forwarding

Secure CoPy (SCP)

- It copies files (and directories) between hosts on a network
- It relies on **ssh** for data transfer, and uses the same authentication of ssh
- The source and target may be
  a. local pathname (absolute or relative paths)
  b. remote host path
- Examples:
  a. ***scp local-file <username>@remote-host: <dest>*** #Copy local-file to remote host
  b. ***scp <username>@remote-host:<remote-file> .*** #Copy remote-file to local host on # the current directory (.)
  c. ***scp -r local-dir <username>@remote-host:dest*** # Copy recursively a directory

Exercise: Move file from/to the unibz cluster

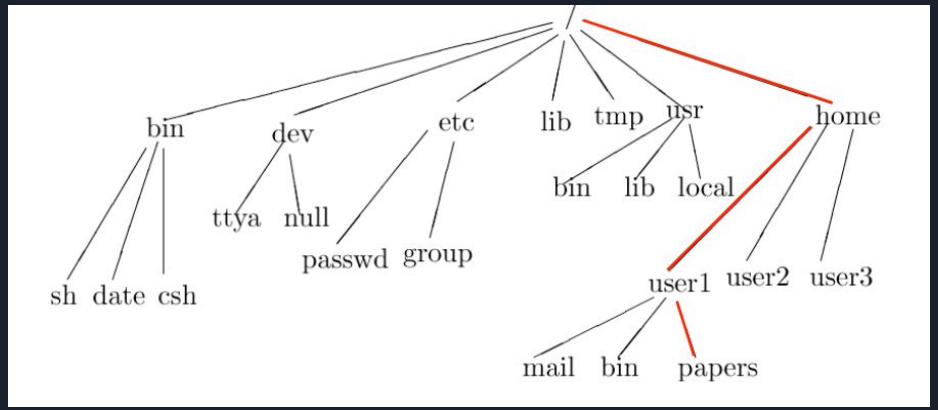
1. Use ***scp*** to copy the file */tmp/testscp* from the login-node to your computer
2. On your computer, create a text file <*name.surname*> and fill it with your name (es. marco.cianfriglia)
3. Copy the <*name.surname*> file from your pc to the home directory on the login-node
4. Use ***ssh*** to execute the following program on the login-node '***cat <name.surname>***' (use your name and surname)

Login-node: ***slurm-ctrl.inf.unibz.it***

Exercise: Move file from/to the unibz cluster Solution

1. Use ***scp*** to copy the file */tmp/testscp* from the login-node to your computer
    a. ***scp gst_mcianfriglia@slurm-ctrl.inf.unibz.it:/tmp/testscp .***
2. On your computer, create a text file <name.surname> and fill it with your name
3. Copy the <name.surname> file from your pc to the home directory on the login-node
    a. ***scp marco.cianfriglia gst_mcianfriglia@slurm.ctrl.inf.unibz.it:***
4. Use ***ssh*** to execute the following program on the login-node 'cat <name.surname>' (use your name and surname)
    a. ***ssh gst_mcianfriglia@slurm-ctrl.inf.unibz.it cat marco.cianfriglia***

Unix file system



Relative path: be careful.
Example: bin ?

- /bin
- /usr/bin
- /home/user1/bin

It depends on your **current working directory**
(pwd or echo $PWD)

- / is called the root directory
- Each file on the file system has a **unique** pathname starting from /
- The path names starting with / are called **absolute paths**, otherwise are called **relative paths**

  /home/user1/papers (absolute)

  user1/papers (relative, starts from /home)
- Two special names:
  - .. parent directory
  - . current directory

Some basic commands

Command name	Usage	Command name	Usage
ls	Prints the list of files and directories	cp	Copies files and directories
mv	Moves files or directories	rm	Removes files and directories
mkdir	Creates a new directory	cd	Changes working directory
rmdir	Removes directories	pwd	Prints current working directory

File system commands example

- Move to */usr* directory (**absolute path**)
  - ***cd /usr***
- From */usr* move to *local/lib* (**relative path**)
  - ***pwd*** #Print working directory
    /usr
  - ***cd local/lib; pwd*** #The ; can be used to sequence of commands on the same line
    /usr/local/lib
- Move back to the *parent* directory (i.e. */usr/local*)
  - ***cd ..*** (Notice the blank space between cd and ..)

Some Unix commands

Command name	Usage	Command name	Usage
file	Prints file type	echo	Prints input string
cat	Prints the content of a file	sort	Order a file
head	Prints the first lines of a file	tail	Prints the last lines of a file
less, more	Prints the content of a file page by page	cut	Selects columns of a file
paste	Merges files by columns	find	Look for files/directories in the filesystem
diff, sdiff	Shows differences between files	grep	Look for a pattern inside a file

Grep

- Look for all the words contain ‘apple’ in */usr/share/dict/words*
  - ***grep apple /usr/share/dict/words***
- Look for all the words starting with ‘apple’ in */usr/share/dict/words*
  - ***grep ^apple /usr/share/dict/words***

. (period character)	Any single character can exist at the specified location
[C1 C2 ..Cn]	C1 or C2 or .. or Cn
[^C1 C2 … Cn]	everything != C1, …, Cn
Pattern*	The pattern 0 or more times
Pattern+	The pattern 1 or more times
^	The beginning of the line
$	The end of the line

stdin, stdout, stderr

Usually a program:

- reads the input from the standard input, ***stdin***
- writes the output on the standard output, ***stdout***
- writes any error on the standard error, ***stderr***

If not specified:

- ***stdin*** = keyboard
- ***stdout*** = ***stderr*** = screen

stdin, stdout, stderr

- It is possible to redirect ***stdin***, ***stdout*** and ***stderr.***

On Bash:

- ‘<’ redirects the stdin
- ‘>’ and ‘2>’ redirect respectively stdout and stderr to a file (if file exists it will be truncated, else it will be created)
- ‘>>’ or ‘2>>’ will append to an existing file
- ‘&>’ redirects both stdout and stderr

stdin, stdout, stderr Examples

- ***ls -l >list.txt*** # The output of 'ls -l' will be stored in the file list.txt
  # If list.txt exists, it will be truncated, otherwise it will be created
- ***ls -l >> list2.txt*** # The output of 'ls -l' will be stored in the file list2.txt
  # If list2.txt exists the new content will be appended at the end of the file
- ***sort < input_file.txt > input_file.sorted***
  sort will read input_file.txt and its output will be saved on input_file.sorted
- ***fakecommand 2> error.out***

Exercise shell

- Redirect on the file *out1.txt* the ***stdout*** of the following command: '***ls -1 ${HOME}***'
- Redirect on the file *err1.txt* the ***stderr*** of the following command: '***fakecmd run***'
- Run the following command and **append** the ***stdout*** on *out1.txt* and redirect the ***stderr*** on *err2.txt*
  ls / mickeymouse
- Print the content of *out1.txt* and *err2.txt* on screen using ***cat***

Exercise shell
Solutions

- Redirect on the file *out1.txt* the ***stdout*** of the following command: '***ls -1 ${HOME}***'
  - ***ls -1 ${HOME} > out1.txt***
- Redirect on the file *err1.txt* the ***stderr*** of the following command: ***'fakecmd run***'
  - ***fakecmd run 2> err1.txt***
- Run the following command and **append** the ***stdout*** on *out1.txt* and redirect ***stderr*** on *err2.txt* : ls / mickeymouse
  - ***ls / mickeymouse >> out1.txt 2>err2.txt***
- Print the content of *out1.txt* and *err2.txt* on screen using ***cat***
  - ***cat out1.txt***
  - ***cat err2.txt***

Exercise shell

- Run the following command and redirect both ***stdout*** and ***stderr*** on *outerr3.txt*
  ls / mickeymouse
- Print the content of *outerr3.txt* using ***less/more***
- Sort the entries in the file *out1.txt* and save the results in file *out1.sorted*
- Create a directory in your home called *output*

Exercise shell Solutions

- Run the following command and redirect both ***stdout*** and ***stderr*** on *outerr3.txt*
  ls / mickeymouse
  - ***ls / mickeymouse &> outerr3.txt***
- Print the content of *outerr3.txt* using ***less/more***
  - ***less outerr3.txt***
  - ***more outerr3.txt***
- Sort the entries in the file *out1.txt* and save the results in file *out1.sorted*
  - ***sort out1.txt > out1.sorted***
- Create a directory in your home called *output*
  - ***mkdir ${HOME}/output***

Exercise shell

- Change your working directory to ***~/output*** **(** ***~*** refers to the home directory)
- Print your current working directory
- Copy *out1.txt out1.sorted* and *out2.txt* into the current working directory (*~/output*)
- Copy the directory *~/output* to your local computer */tmp* folder (hint: remember ***scp***)

Exercise shell
Solutions

- Change your working directory to *~/output* **(~** refers to the home directory)
  - ***cd ${HOME}/output*** **or** ***cd ~/output***
- Print your current working directory
  - ***pwd***
- Copy *out1.txt out1.sorted* and *out2.txt* into the current working directory (*~/output*)
  - ***cp ${HOME}/out1.txt ${HOME}/out1.sorted .***
  - ***cp ${HOME}/out1.txt ${HOME}/out1.sorted ${HOME}/output***
- Copy the directory *~/output* to your local computer */tmp* folder (hint: remember ***scp***)
  - ***scp -r <username>@slurm-ctrl.inf.unibz.it:output /tmp***

The pipe |

- A pipe '|' allows to concatenate commands by redirecting the ***stdout*** of a command as the ***stdin*** of the next command.
- It is a simple yet powerful tool that allows to solve efficiently many tasks

List in alphabetical order all the different words of a text, with the number of occurrences next to them, in order of frequency (first the most frequent words).

- ***tr -s ' ' '\n' < text | sort | uniq -c | sort -n -r***

N.B. This is more efficient than executing each command individually . Do you know why?

Exercise pipe

- Print the files and directory of the */tmp* folder on the login-node, one record per line (hint: see man ls), sort the records lexicographically and store the result on *~/output/tmp.sorted*

  - ***ls -1 /tmp | sort > ~/output/tmp.sorted***

Linux file permissions

ls -la

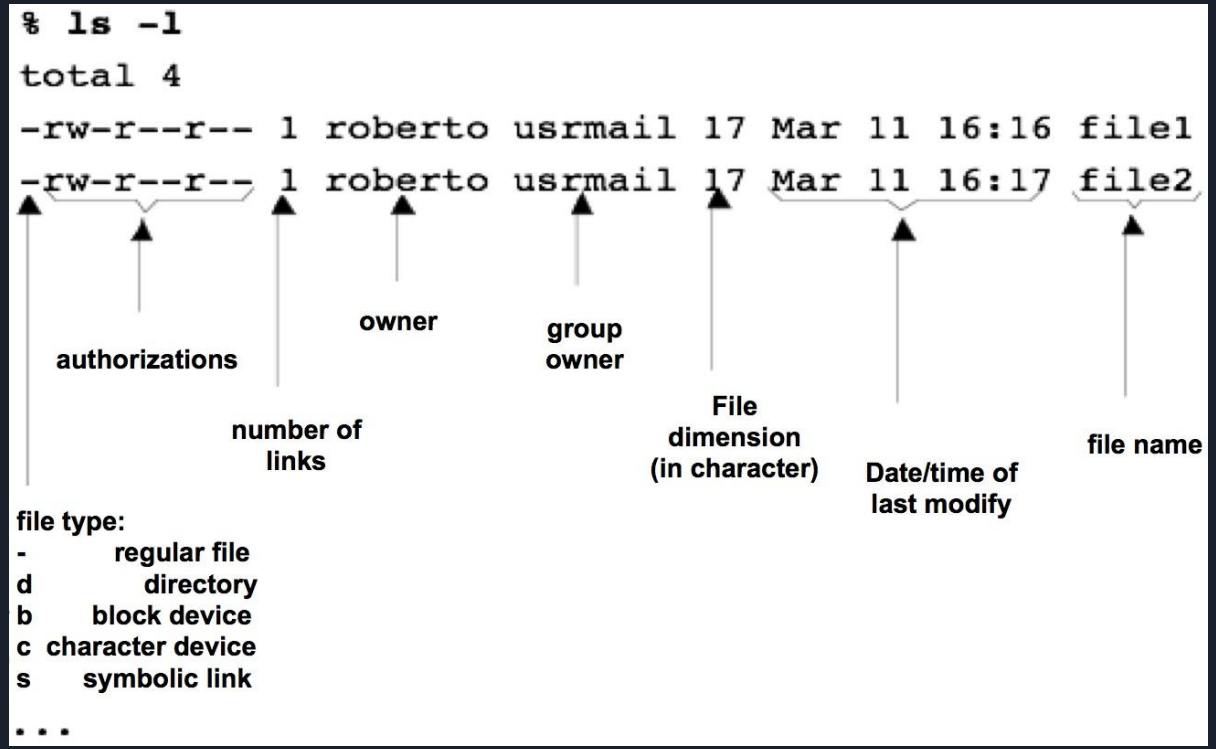
-rwxrw-r-x user group filename
drwx---r-x user group directory

- What does it mean?
  r=read, w=write, x=execute (regular files)
  r=read, w=create/remove, x=search (directories)
  r=read, w=write, x=n/a (special files, like device files)

- How to change files and directories permissions?
  chmod [OPTION] MODE/OCTAL-MODE FILE …
  What does the following command *chmod 753 file* ?

Octal	Binary	Permission
0 1 2 3	000 001 010 011	--- --x -w- -wx
4 5 6 7	100 101 101 111	r-- r-x rw- rwx

Linux File Permissions



Exercise command line

- List the permissions of the file *~/output/out1.txt*
- Set the permissions for *~/output/out1.txt* to **rwx** for the owner, ***rw*** for the group and **zero permission** for other
- Set the permissions for the *~/output* directory to ***r-- rw-rw-*** . Can you list the content of the directory? Can you access the content of its files?
- Set the permissions for the *~/output* directory to ***--x rw-rw-*** . Can you list the content of the directory? Can you access the content of its files?

Exercise command line

- List the permissions of the file *~/output/out1.txt*
  - ***ls -l ~/output/out1.txt***
- Set the permissions for *~/output/out1.txt* to **rwx** for the owner, **rw** for the group and **zero permission** for other
  - ***chmod 760 ~/output/out1.txt***
- Set the permissions for the *~/output* directory to ***r-- rw-rw-*** . Can you list the content of the directory? Can you access the content of its files?
  - ***chmod 566 ~/output;*** Yes, but error; No
- Set the permissions for the *~/output* directory to ***--x rw-rw-*** . Can you list the content of the directory? Can you access the content of its files?
  - ***chmod 166 ~/output; No; Yes***

Bash and scripting

Some BASH important files

- When an interactive shell starts it reads and executes the ***~/.bashrc*** config file

- When bash is invoked as an interactive login shell it reads and executes commands from the files (if they exist)
  - /etc/profile
  - ~/.bash_profile
  - ~/.bash_login
  - ~/.profile

- When a login shell exits, bash reads and executes commands from the file ~/.bash_logout, if it exists.
- The commands history can be found in ~/.bash_history
  - Command history displays the content of ~/.bash_history

vi Editor

vi is a command line editor, available on Linux (vim the enhanced version)

Two operative modes:

- *Command-mode*
- *Input-mode*

It starts in *command-mode*; you may switch from *command* to *input* modes by using several commands (see next slides)

To exit *input-mode*, use the Esc button (<ESC>)

Once in *input-mode*, any character you insert will be treated as text and it will be added to your file.

vi editor minimal cheetsheet

Input Commands
(end with <ESC>)

a	Append after cursor
A	Append after line
i	Insert before cursor
I	Insert before line
o	Open line below
O	Open line above
r	Replace one character
R	Replace many characters
p	Put after
P	Put before

File Management Commands
(please note the ':' before commands)

:w	Store the file content
:wq	Store the file content and quit
:x	Same as :wq
:q!	Quit without saving changes

Auxiliary commands

x	Delete character to right of cursor
X	Delete character to left of cursor
D	Delete the rest of the line
dd	Delete current line
y	Copy the current line
u	Undo last change

vi Example

From your terminal

- Launch vi to start writing a new file
  vi my_file

From vi

- Switch from *command-mode* to *input-mode*
  Press i
- Write your name
- Come back to *command-mode*, save the content and quit vi
  a. Press <Esc> to exit *input-mode*
  b. :wq or :x or :w and :q to save changes and quit

Bash scripting

It is possible to automate some operations by using a shell script

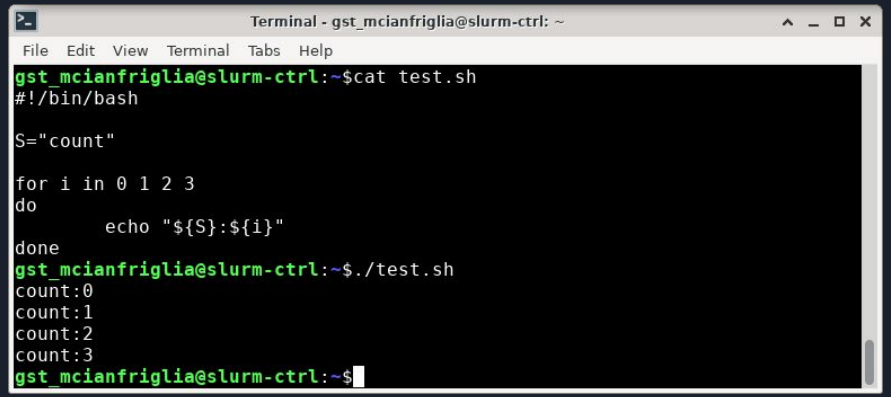
Example: test.sh

```
#!/bin/bash

S=”count”
for i in 0 1 2 3
do
        echo “${S}: ${i}”
done
```

In order to execute it

1. ***chmod u+x test.sh*** # add execution permission to the owner
2. ***./test.sh*** # execute it



Bash scripting

Example: test.sh

```
#!/bin/bash

S=”count”
for i in 0 1 2 3
do
        echo “${S}: ${i}”
done
```

- #! **shebang**: is used to tell the operating system which interpreter to use to parse the rest of the file
- ***S*** is a variable
- ***for do … done*** defines a loop
- *i* is the variable that counts the loop iterations
- ***echo***: a shell command that takes as input a string and it prints it to the screen

Bash Variables

- Variables allow programmers to store data, alter and reuse them throughout the script
- Any variable can get any kind of data, basically strings or numbers
- The name of the variable is the string preceding the =
- To access the value of a variable use the $

```
S=”summer school”        #Assing the string “summer school” to S
echo “${S}”              # Echo print the value of S
```

Bash Built-in variables

- $# contains the number of input parameters (like argc)
- $0 contains the script name
- $j with 1 <= j<= $# are the input parameters
- $? the exit code of the last executed command
- $IFS Internal Field separator (default space, tab, and newline)

Bash if-then-else

```
if [ TEST ]
then
        <do some stuff>
elif [ TEST2 ]
then
        <do some stuff-2>
else
        <do some stuff-3]
fi
```

- The *if-then-else* statement starts with ***if*** keyword, followed by a conditional expression and the keyword ***then***.
- The statement ends with the keyword ***fi***
- The ***else /elif*** statements are optionals
- The ***elif*** is followed by a conditional expression and the keyword ***then***
- It is possible to have nested if statements

Common test operator

INT1 -eq INT2: True if INT1== INT2

INT1 -ne INT2: True if INT1 != INT2

INT1 -gt INT2: True if INT1 > INT2

INT1 -lt INT2: True if INT1 < INT2

INT1 -ge INT2: True if INT1 >= INT2

INT1 -le INT2: True if INT1 <= INT2

STR1 = STR2: True if STR1 is equal to STR2

STR1 != STR2: True if STR1 is not equal to STR2

-d FILE: True if FILE exists and is a directory

-e FILE: True if FILE exists and is a file

-f FILE: True if FILE exists and is a regular file

-r FILE: True if FILE exists and is readable

-w FILE: True if FILE exists and is writable

-x FILE: True if FILE exists and is executable

-n VAR: True if the length of VAR is > 0

-z VAR: True if the VAR is empty

if-then-else examples

```
#!/bin/bash
#This is a comment
ARGC=2
if [ $# -ne ${ARGC} ]
then
        echo “Usage $0: <src> <dst>”
        exit 1
fi
cp $1 $2
exit $?
```

```
#!/bin/bash
ARGC=1
USER=”test-user”
if [ $# -ne ${ARGC} ]
then
        echo “Usage $0: <user>”
        exit 1
fi

if [ “${USER}” == “${1}” ]
then
        echo “Hi ${USER}”
else
        echo “Bye”
fi
```

Bash array

- An array is a variable containing multiple values.
- Any variable may be used as an array.
- There is no maximum limit to the size of an array, nor any requirement that member variables be indexed or assigned contiguously.
- Arrays are zero-based: the first element is indexed with the number 0

Bash array

Multiple declaration options:

- Indirect declaration: ARRAY[idx]=value, idx is treated as an arithmetic expression that must evaluate to a positive number
- Explicit declaration : declare -a ARRAY (declare shell builtins)
- Compound assignments: ARRAY=(value1 value2 … valueN)
- Dereferencing the variables in an array:
  - echo ${ARRAY[*]} # print all the array values * or @ have the same effect. NB the {} are necessary
  - echo ${ARRAY[2]} # print the second element of the array

Bash for loops

```
# Explicit range
for i in 1 2 … N
do
        <do stuff on i>
done
```

```
# Range border
for i in {1..100}
do
        <do stuff on i>
done
```

```
# Iterates over cmd-output
for r in $(Unix/Linux cmd output)
do

        <do stuff on r>
done
```

```
# Explicit range
for f in file1 file2 ...fileN
do
        <do stuff on f>
done
```

```
# For-loops and array
F=(“file1” “file2” … “fileN”)
for r in ${F[@]}
do

        <do stuff on r>
done
```

Bash while loop

```
while [ condition]
do
        <do some stuff>
done
```

```
#!/bin/bash

x=0
while [ $x -lt 10 ]
do
        echo “${x}”
        x=$(( $x + 1 ))
done
```

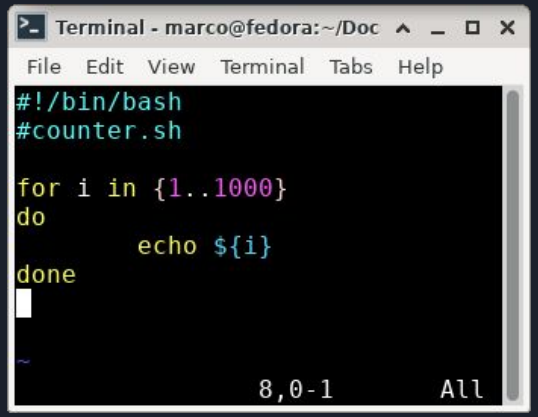
- To iterate over variable, for loops are more convenient
  No need to update the counter
- While loops usually used to read data line by line from a file

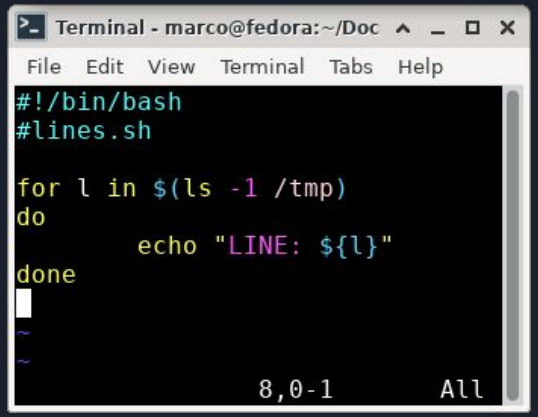
```
#!/bin/bash
...
INPUT_FILE=”/etc/passwd”
while read -r line
do
        echo $line
done < “${INPUT_FILE}”
```

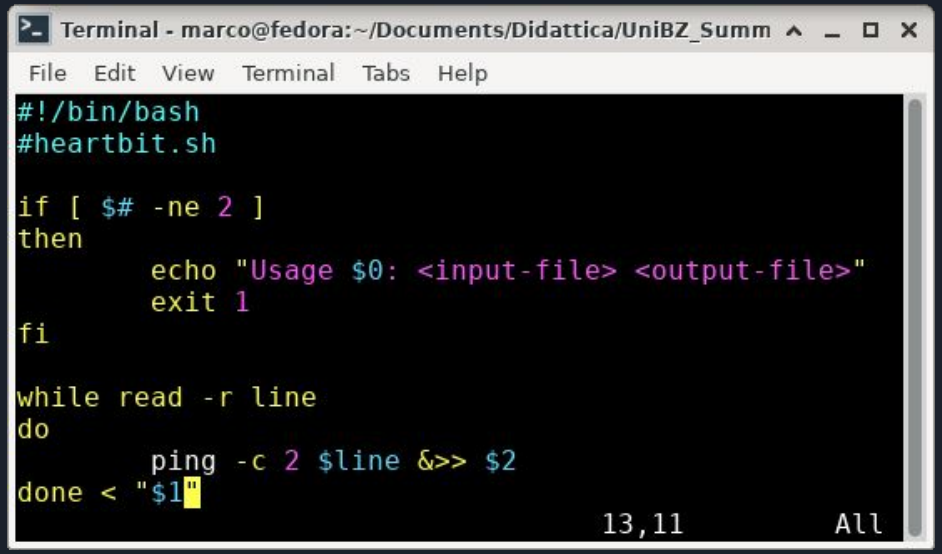
Exercise

- Write a Bash script (***counter.sh***) that prints all the number from 0 to 1000
- Write a Bash script (***lines.sh***) that reads the output of ‘ls -1 /tmp’ and add the prefix “LINE:” to each line
- Write a Bash script (***heartbit.sh***) that takes as input a file containing a list of ip addresses, one per line and the name of the output file
  For each ip address, run the command “ping -c 2” and save its output and error by appending it to the output file.
- Write a Bash script (***compareFiles.sh***) that takes as input two files and returns the one that has more lines

Exercise Solution







Exercise Solution

Terminal - marco@fedora:~/Documents/Didattica/UniBZ_SummerSchoolHPC/test

File Edit View Terminal Tabs Help

```
#!/bin/bash
#compareFiles.sh

if [ $# -ne 2 ]
then
        echo "Usage $0: <file1> <file2>"
        exit 1
fi

if ! [ -f "$1" ]
then
        echo "Error $1 is not a regular file"
        exit 1
fi
if ! [ -f "$2" ]
then
        echo "Error $2 is not a regular file"
        exit 1
fi

c1=0
while read -r line
do
        c1=$(( $c1 + 1 ))
done < "$1"
echo "$1 has $c1 lines"

c2=0
while read -r line
do
        c2=$(( $c2 + 1 ))
done <"$2"
echo "$2 has $c2 lines"

if [ $c1 -gt $c2 ]
then
        echo "$1 WIN"
else
        echo "$2 WIN"
fi
~
                                                    39,14-21      All
```

Terminal - marco@fedora:~/Documents/Didattica/UniBZ_SummerSchoolHPC/test

File Edit View Terminal Tabs Help

```
#!/bin/bash
#compareFiles2.sh

if [ $# -ne 2 ]
then
        echo "Usage $0: <file1> <file2>"
        exit 1
fi

if ! [ -f "$1" ]
then
        echo "Error $1 is not a regular file"
        exit 1
fi
if ! [ -f "$2" ]
then
        echo "Error $2 is not a regular file"
        exit 1
fi

c1=$(wc -l "$1" | cut -d' ' -f 1)
echo "$1 has $c1 lines"

c2=$(wc -l "$2" | cut -d' ' -f 1)
echo "$2 has $c2 lines"

if [ $c1 -gt $c2 ]
then
        echo "$1 WIN"
else
        echo "$2 WIN"
fi
~
~
~
~
~
~
~
~
~
"compareFiles2.sh" 32L, 428B written                2,14          All
```