

Consortium des Équipements de Calcul Intensif

Data storage on the CECI clusters

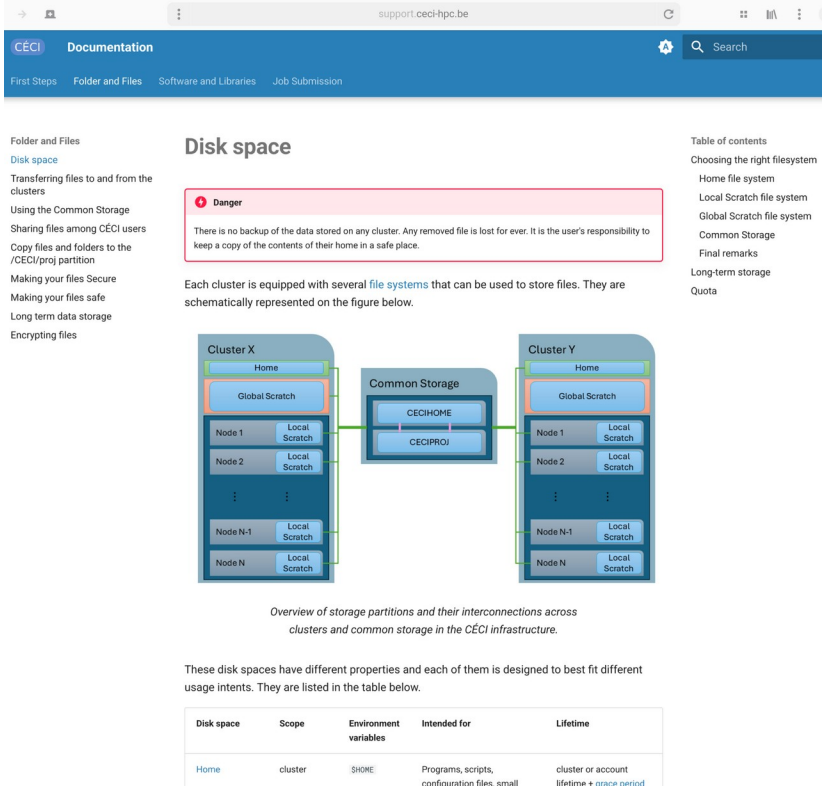
Ariel Lozano

CÉCI HPC Training
19 Nov 2026

Outline

https://support.ceci-hpc.be/doc/ManagingFiles/Storage/

https://support.ceci-hpc.be/doc/SubmittingJobs/SlurmJobFilesystems/



support.ceci-hpc.be

CÉCI Documentation

Search

First Steps | Folder and Files | Software and Libraries | Job Submission

Folder and Files
- Disk space
- Transferring files to and from the clusters
- Using the Common Storage
- Sharing files among CÉCI users
- Copy files and folders to the /CECI/proj partition
- Making your files Secure
- Making your files safe
- Long term data storage
- Encrypting files

Disk space

Danger

There is no backup of the data stored on any cluster. Any removed file is lost for ever. It is the user's responsibility to keep a copy of the contents of their home in a safe place.

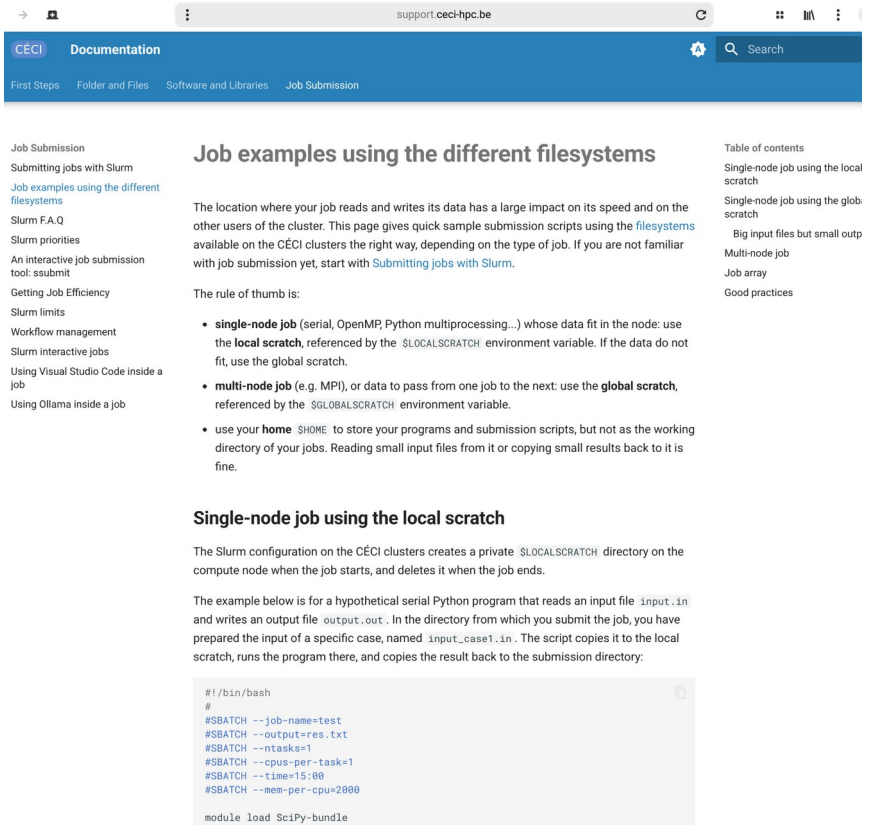
Each cluster is equipped with several file systems that can be used to store files. They are schematically represented on the figure below.

Overview of storage partitions and their interconnections across clusters and common storage in the CÉCI infrastructure.

These disk spaces have different properties and each of them is designed to best fit different usage intents. They are listed in the table below.

Disk space	Scope	Environment variables	Intended for	Lifetime
Home	cluster	$HOME	Programs, scripts, configuration files, small	cluster or account lifetime + grace period

Table of contents
- Choosing the right filesystem
  - Home file system
  - Local Scratch file system
  - Global Scratch file system
  - Common Storage
  - Final remarks
- Long-term storage
- Quota



support.ceci-hpc.be

CÉCI Documentation

Search

First Steps | Folder and Files | Software and Libraries | Job Submission

Job Submission
- Submitting jobs with Slurm
- Job examples using the different filesystems
- Slurm F.A.Q
- Slurm priorities
- An interactive job submission tool: submit
- Getting Job Efficiency
- Slurm limits
- Workflow management
- Slurm interactive jobs
- Using Visual Studio Code inside a job
- Using Ollama inside a job

Job examples using the different filesystems

The location where your job reads and writes its data has a large impact on its speed and on the other users of the cluster. This page gives quick sample submission scripts using the filesystems available on the CÉCI clusters the right way, depending on the type of job. If you are not familiar with job submission yet, start with Submitting jobs with Slurm.

The rule of thumb is:

- **single-node job** (serial, OpenMP, Python multiprocessing...) whose data fit in the node: use the **local scratch**, referenced by the `$LOCALSCRATCH` environment variable. If the data do not fit, use the global scratch.
- **multi-node job** (e.g. MPI), or data to pass from one job to the next: use the **global scratch**, referenced by the `$GLOBALSCRATCH` environment variable.
- use your **home** `$HOME` to store your programs and submission scripts, but not as the working directory of your jobs. Reading small input files from it or copying small results back to it is fine.

Single-node job using the local scratch

The Slurm configuration on the CÉCI clusters creates a private `$LOCALSCRATCH` directory on the compute node when the job starts, and deletes it when the job ends.

The example below is for a hypothetical serial Python program that reads an input file `input.in` and writes an output file `output.out`. In the directory from which you submit the job, you have prepared the input of a specific case, named `input_case1.in`. The script copies it to the local scratch, runs the program there, and copies the result back to the submission directory:

```
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=res.txt
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --time=15:00
#SBATCH --mem-per-cpu=2000

module load SciPy-bundle
```

Table of contents
- Single-node job using the local scratch
- Single-node job using the glob scratch
  - Big input files but small outp
- Multi-node job
- Job array
- Good practices

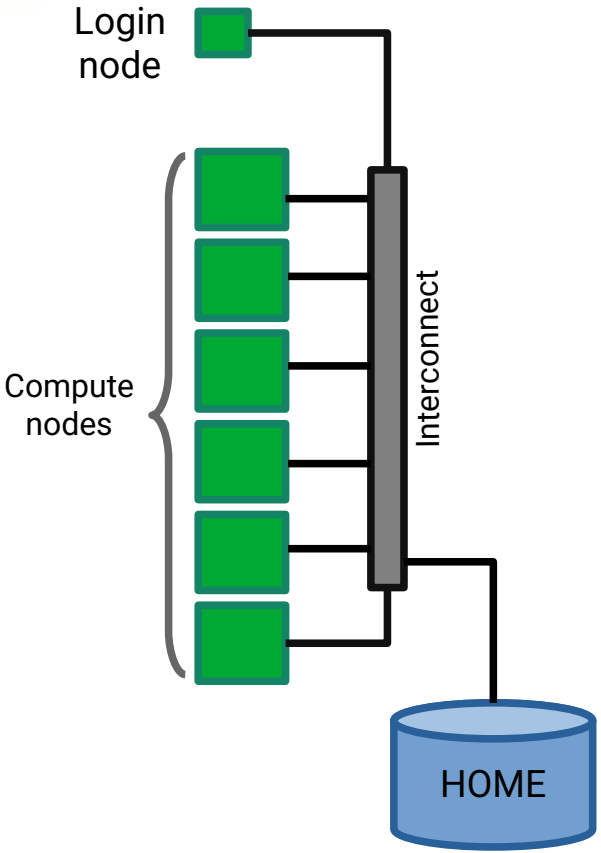
DISCLOSURE



WARNING: No data on the CECI clusters has backups

You are responsible of copying over your useful data you need to store long term somewhere else

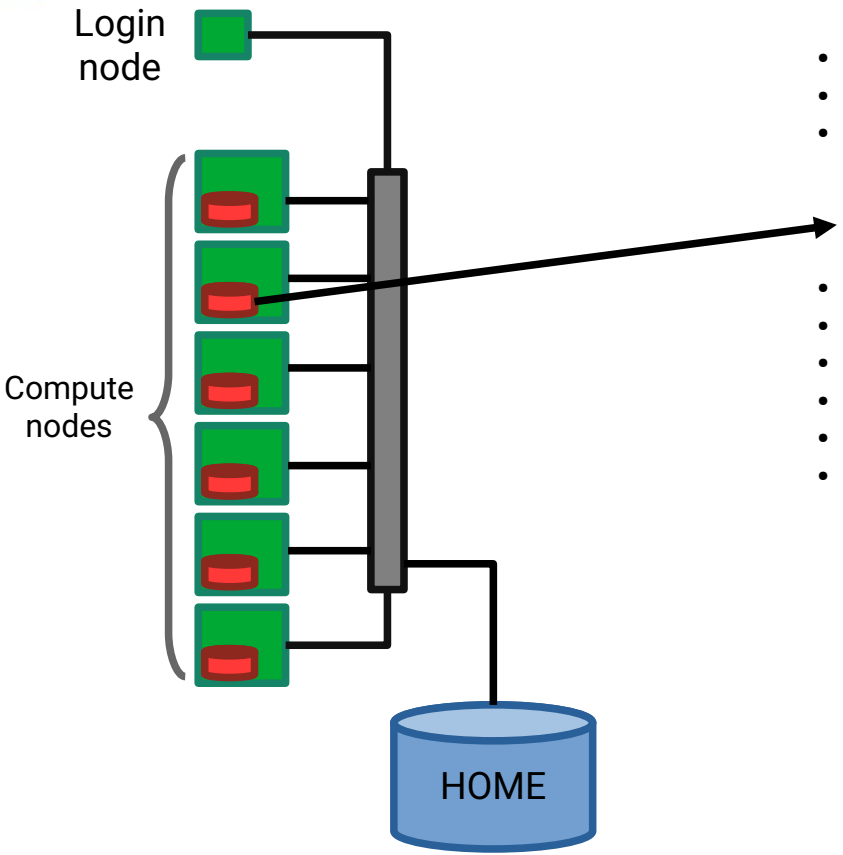
Filesystems on CECI clusters



`$HOME`

- Area where you land on login
- Network storage accessible by all compute nodes and login
- In general a small quota available (100GB)

Filesystems on CECI clusters



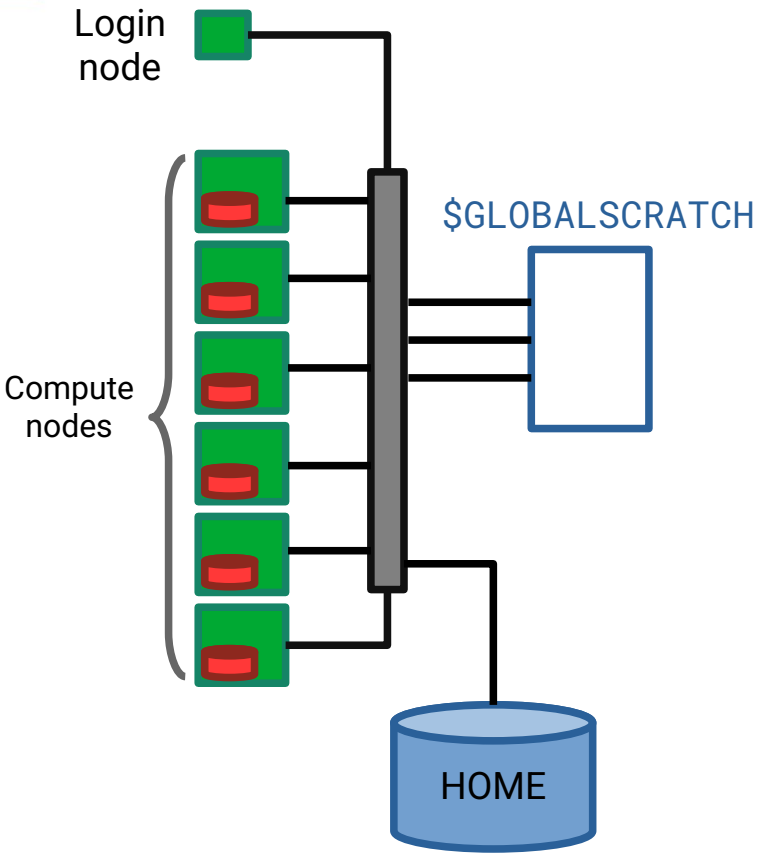
`$HOME`

- Area where you land on login
- Network storage accessible by all compute nodes and login
- In general a small quota available (100GB)

`$LOCALSCRATCH`

- Internal high performance storage inside each compute node
- Accessible **only** from the compute node running your job
- Shared with other jobs running on the same node
- Usage only temporary during the job runtime
- Serial jobs or Parallel SMP (openmp, python multiprocessing, ...)
- Data is **automatically purged** when job ends

Filesystems on CECI clusters



$HOME

- Area where you land on login
- Network storage accessible by all compute nodes and login
- In general a small quota available (100GB)

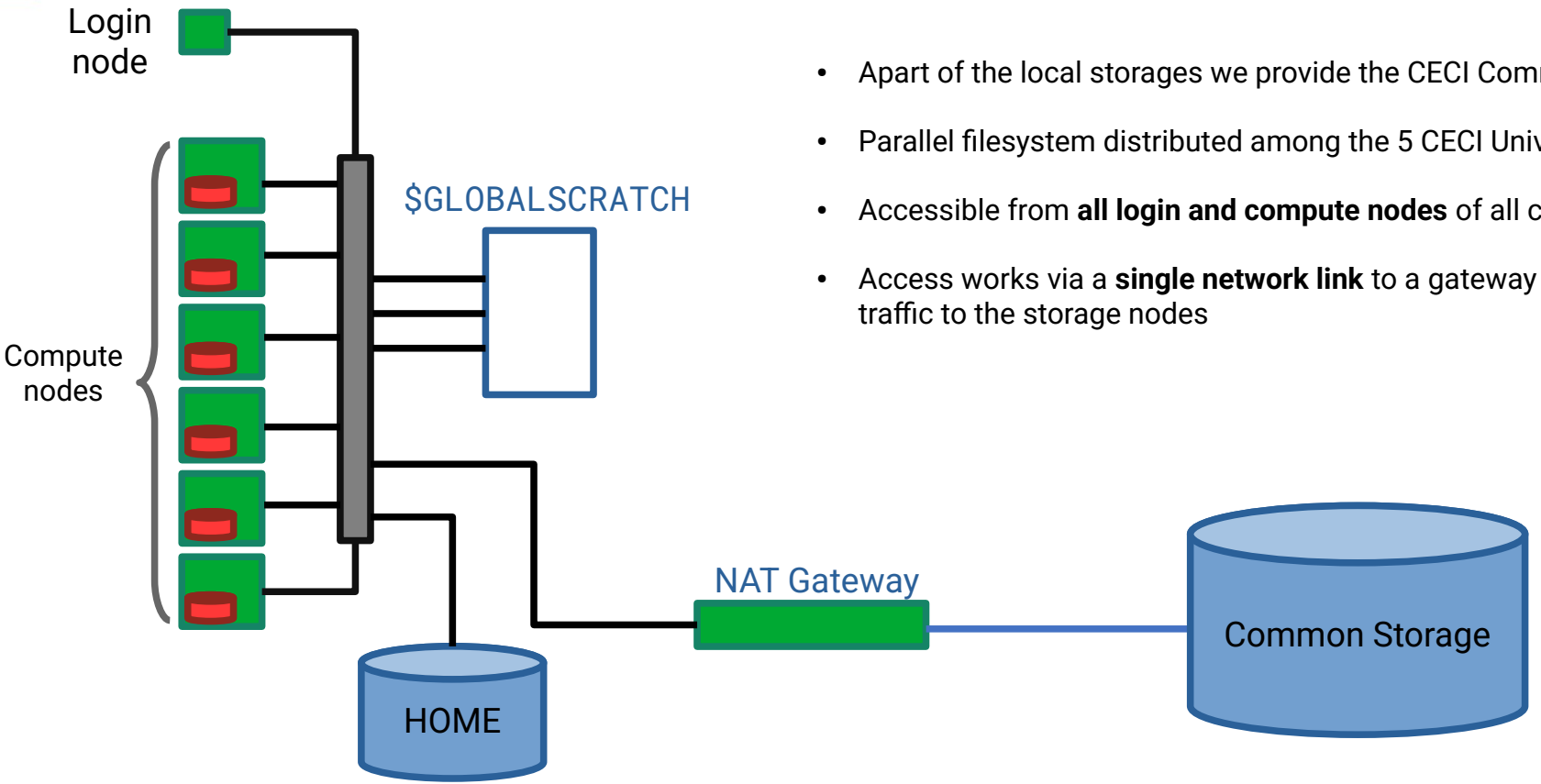
$LOCALSCRATCH

- Internal high performance storage inside each compute node
- Accessible **only** from the compute node running your job
- Shared with other jobs running on the same node
- Usage only temporary during the job runtime
- Serial jobs or Parallel SMP (openmp, python multiprocessing, ...)
- Data is **automatically purged** when job ends

$GLOBALSCRATCH

- Based on a distributed filesystem
- Network storage accessible by all compute nodes and login
- All jobs but **only option** for **multinode-parallel** jobs (big MPI jobs)
- Data there stays persistently (but can be removed in yearly maintenances)
- Please cleanup from time to time

CECI Common storage

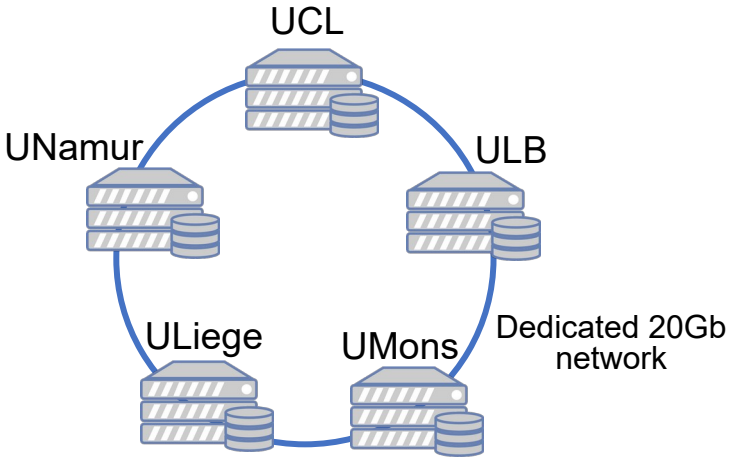


- Apart of the local storages we provide the CECI Common Storage
- Parallel filesystem distributed among the 5 CECI Universities
- Accessible from **all login and compute nodes** of all clusters
- Access works via a **single network link** to a gateway server routing the traffic to the storage nodes

CECI Common storage

Hardware setup

- 4x storage nodes in each university datacenter
- Dedicated 20Gbps network backbone
- Parallel filesystem based on the CEPH solution
- Total net capacity of 1.2 PB
- Same unique filesystem available from all the clusters



Split into two distinct areas

`/CECI/home`

- With your CECI account you get a folder referenced via the `$CECIHOME` variable
- Quota of 100GB and 100k files
- Transfer data among clusters
- Store common tools, submission scripts, small input data, code source, binaries

`/CECI/proj`

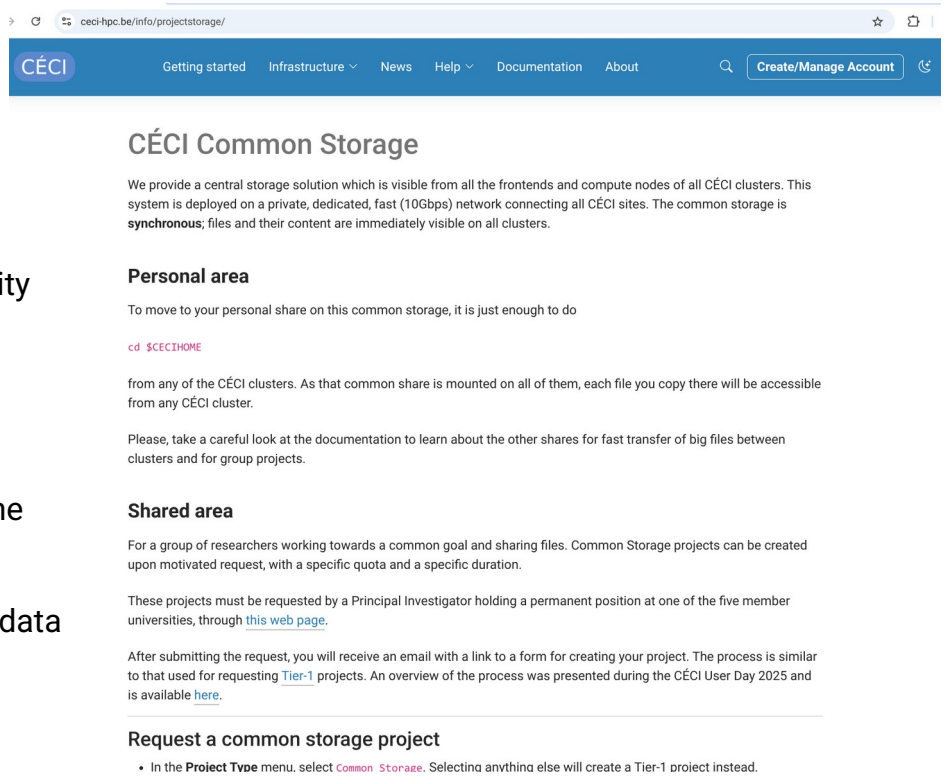
- Area available for Common Storage project requests
- Each univerisity can clain up-to 200TB for **all** its projects
- Transfer data among clusters $GLOBALSCRATCH
- Store output data to be processed later, big input data to be used in the different clusters

CECI Common storage: Projects

`/CECI/proj/<project_acronym>`

- Area dedicated for groups of researchers working towards a common goal and sharing files
- Common Storage projects are created upon request of a **Principal Investigator** only
- In the request it must be specified and **justified** the desired total capacity of the volume
- Everybody in the group will get read/write access to the full project folder
- By default the project folder is pinned to a metadata server of your home institution.
  If the group will work mainly on a CECI cluster from other institution, you can request to pin the project or a subdirectory to the closest metadata server

https://www.ceci-hpc.be/info/projectstorage/



CECI Common storage: Remarks

- Shared by all the clusters, but **is not** a scratch space. Due to the distributed infra slower than $LOCALSCRATCH and $GLOBALSCRATCH.
- Not a long-term archive: the data is replicated against disk failure, but there is no backup.
- Used correctly, it is the fastest way to move data between clusters.
- Avoid lots of small files: pack them into one file.
  - In your jobs: copy to $LOCALSCRATCH and unpack when the job starts, and pack the results into a file before copying back
  - Example: For a myfilte.tar.gz of 6.3 GB containing 50000 files (NIC5)

	As a single archive	**Copying the files one by one**
Common Storage → $LOCALSCRATCH	47s	14m 26s
$LOCALSCRATCH → Common Storage	31s	1m 37s

How much data can I store on each fs?

Cluster	$HOME	$CECIHOME	$LOCALSCRATCH (hardware support)	$GLOBALSCRATCH (Total Capacity - fstype)
NIC6	100 GB	100 GB	3.6TBGB (local SSD)	5TB (800TB - beegfs)
Dragon3	80 GB	100 GB	7TB (local SSD)	XX[1] (100TB - beegfs)
Lyra	100 GB	100 GB	2TB (local SSD)	5TB (500TB - CEPH)
Lemaitre4	100 GB	100 GB	180GB (local SSD)	XX[1] (320TB - beegfs)
Hercules[*]	2TB	100GB	her2-w065..096: 1TB her2-w099..126: 4TB her2-w127..128: 8TB (HDD RAID0)	2TB (250TB - CEPH)

[1] No limits enforced but remember resources are shared among all ~800 CECI users!!
[*] Hercules HOME is a special case with high quota
https://support.ceci-hpc.be/doc/ManagingFiles/Storage/#quota

How much am I using?

Use the `ceci-quota` command on any cluster

```
 [myuser@lyra-l02: ~]---> $ ceci-quota

 Diskquotas for user myuser
Filesystem           used      limit      files      limit
$HOME            39.9 GiB  100.0 GiB      50113     100000
$GLOBALSCRATCH    1.5 TiB    5.0 TiB       4504     300000
$CECIHOME        11.4 GiB  100.0 GiB       4390     100000
```

```
 [myuser@lm4-f001: ~]---> $ ceci-quota

 Diskquotas for user myuser
Filesystem       used      limit      files      limit
$HOME           4.14G       100G      3.82K
$GLOBALSCRATCH 4.3 GB  unlimited          8  unlimited
$CECIHOME    11.4 GiB  100.0 GiB       4390     100000
```

Filesystems on CECI clusters

Wrap-up

- $LOCALSCRATCH will be always the ideal for single core jobs
- If all your input/output data volume for your job fits on $LOCALSCRATCH **use it** your code will speed up just for doing that
- $GLOBALSCRATCH is the only choice for parallel multi-node jobs
- $GLOBALSCRATCH can also be used to store big input data, or recover useful output for a job working on $LOCALSCRATCH
- There's **no** reason to use $HOME for job I/O. Perfomance is the **lowest** and the usable space the **smallest**

Jobs submission

How can we control the data location from a Slurm job?

Making use of the pre-defined environment variables:

`$HOME`

`$LOCALSCRATCH`

`$GLOBALSCRATCH`

`$CECIHOME`

Extra useful variables defined on-the-fly inside a job environement:

`$SLURM_JOB_ID` the Job ID value
`$SLURM_SUBMIT_DIR` directory where the job was submitted from

Jobs submission: single node localscratch

If all input and output fits on the $HOME and $LOCALSCRATCH of the cluster

```
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=res.txt
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --time=15:00
#SBATCH --mem-per-cpu=2000

module load SciPy-bundle

# 1. copy the input file to the local scratch and move there
cp ${SLURM_SUBMIT_DIR}/input_case1.in ${LOCALSCRATCH}/input.in
cd ${LOCALSCRATCH}

# 2. run the program from $HOME, only I/O on the local scratch
srun python ${HOME}/mycode/mycode.py

# 3. copy the results back to the submission directory
cp output.out ${SLURM_SUBMIT_DIR}/
```

- Python code that requires just an `input.in` file
- The code is located on a folder inside my $HOME
- You prepare the `input_case1.in` for a given case in a subfolder in your $HOME, you locate also the slurm submission script as shown on that folder
- The useful output.out file produced must be copied back to the submission directory

Jobs submission: single node globalscratch

If the input and output data are too big for both home and local scratch

```
$ mkdir -p $GLOBALSCRATCH/myjob1000
$ cd $GLOBALSCRATCH/myjob1000

> create the submission script myjob.sh as

#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=res.txt
#SBATCH --ntasks=1
#SBATCH --time=15:00
#SBATCH –mem-per-cpu=2000

module load SciPy-bundle

# this job starts in a directory from $GLOBALSCRATCH
# the input is already there, output will be written there too

srun python ${HOME}/mycode/mycode.py
```

- Python code that requires just an `input.in` file and produces and output.out file
- Prepare the input of your job in a directory inside $GLOBALSCRATCH (wget, scp, rsync, ...)
- The submission script is located in the same directory and it will be just the execution of the code
- When the job finishes you will have the results on the same directory

Jobs submission: single node mix

You can consider a mix if the input data is too big, but the produced output fits on LOCALSCRATCH

```
$ mkdir -p $GLOBALSCRATCH/myjob1000
$ cd $GLOBALSCRATCH/myjob1000

> create the submission script myjob.sh as

#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=res.txt
#SBATCH --ntasks=1
#SBATCH --time=15:00
#SBATCH –mem-per-cpu=2000

module load SciPy-bundle

# work in the local scratch
cd ${LOCALSCRATCH}

srun python ${HOME}/mycode/mycode.py ${SLURM_SUBMIT_DIR}/input.in

# copy the useful output back before the end of the job
cp output.out ${SLURM_SUBMIT_DIR}/

> and we can submit it directly from here
$ sbatch myjob.sh
```

- Prepare the input of your job in a directory inside $GLOBALSCRATCH (wget, scp, rsync, ...)
- To this code we can provide the input file as an argument. We move inside $LOCALSCRATCH and via the $SLURM_SUBMIT_DIR variable we can point to the input on the global scratch
- The useful output must be copied back to the submission directory

Jobs submission: multi-node job

In this case always use $GLOBALSCRATCH for all the job I/O

```
$ mkdir myjob1000
$ myjob1000

> create the submission script myjob.sh as

#!/bin/bash
#
#SBATCH --job-name=test_mpi
#SBATCH --output=res_mpi.txt
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --time=60:00
#SBATCH --mem-per-cpu=1000

module load OpenMPI

# one work directory on the global scratch
RUNDIR=${GLOBALSCRATCH}/mpijob_${SLURM_JOB_ID}
mkdir -p ${RUNDIR}
cp ${SLURM_SUBMIT_DIR}/input.in ${RUNDIR}/
cd ${RUNDIR}

srun ${HOME}/my_mpi_program/run.x

> and we can submit it directly from here
$ sbatch myjob.sh
```

- If the input fits in the $HOME we can prepare it there together with the submission script
- In the job script, we create a work directory inside $GLOBALSCRATCH and copy over the input file there. The $SLURM_JOB_ID variable will make this directory unique.
- We move into this directory and execute the program
- When the job finishes you will have the results on the same directory

Jobs submission: packing a results folder

- Filesystems are not friendly with managing lot of files, it adds a lot of overhead to read/write operations
- This is specially worst for distributed filesystems (global scratch, Common Storage)
- If you have a single node job producing results with lot of files (~2k or more), if they fit you must use the $LOCALSRATCH for I/O and pack the folder before copying it out

```
> a line shown in the examples as

# copy the useful output back before the end of the job
cp output.out ${SLURM_SUBMIT_DIR}/


> would become something as

# pack the whole content of the local scratch into a single file in the submission dir
tar czf results_${SLURM_JOB_ID}.tar.gz -C ${LOCALSCRATCH} ./
cp results_${SLURM_JOB_ID}.tar.gz ${SLURM_SUBMIT_DIR}/
```

To wrap up

Never do direct I/O on your `$HOME`

Prioritize the usage of `$LOCALSCRATCH` if your jobs allow it (e.g. jobs running on a single node)
Remember this area is shared with other users of the node and there's no quota!!

Never redirect outputs to -> `/tmp` use always `$LOCALSCRATCH` instead

For your multi-node parallel jobs **always** use `$GLOBALSCRATCH` **never** your `$HOME`



Remember to backup your useful data somewhere else outside the clusters

Exercise ....

Check the exercise to experiment with, available from any clusters at:

```
cat /CECI/proj/training/ceci_storages/README.md
```