MÉSOCENTRE
UNIVERSITÉ
Clermont Auvergne

AuBi
Auvergne Bioinformatique

Utilisation d'un Cluster de Calcul

2026

Mésocentre – Plateforme AuBi

UNIVERSITÉ Clermont Auvergne

Pré-requis

- Savoir se connecter au cluster de calcul HPC2
- Connaissance de base de la ligne de commande (cut, grep, sort, ...)
- Transfert et (dé)compression de fichiers

Objectifs

- Connaissance des concepts et bonnes pratiques d'utilisation des ressources du cluster de calcul
- Soumission et suivi de jobs
- Devenir autonome dans l'usage des ressources de calcul

Programme

- Commandes Linux, quelques rappels ?
- Environnement de travail
  - + infrastructure d'un cluster de calcul
  - + outils
  - + TP1
- SLURM
  - + commandes de base
  - + jobs
  - + monitoring
  - + TP2
- SLURM avancé
  - + job array
  - + compression / décompression
  - + multi-threading
  - + TP3

Pré-requis : Connexion au frontal HPC2

- Avoir un compte au Mésocentre Clermont-Auvergne
- Avoir renseigné sa clé publique sur hub.mesocentre.uca.fr
- Se connecter au serveur de calcul HPC2

$ ssh *<login>*@hpc2.mesocentre.uca.fr

où *<login>* : votre identifiant utilisateur
et hpc2.mesocentre.uca.fr : adresse du serveur

Exemple : syldubo@hpc2.mesocentre.uca.fr

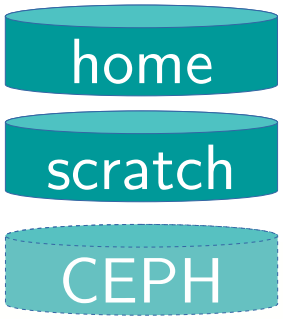
Connexion au frontal HPC2

- Accès à un environnement de travail

  + ressources mutualisées (CPU, Mémoire, espace disque)
  + outils / logiciels en partage
  + accès aux ressources de calcul dédiées (ex. nœuds GDEC, LMGE, ...)
  + accès aux espaces de stockage dédiés (ex. iGReD, PIAF, UNH)

Rappel : Mésocentre Clermont-Auvergne

- Infrastructure



100 Go

500 Go

xx To (données en duplicats, triplicats)

- Portail du Mésocentre

URLs : https://portail.dsi.uca.fr : accès au suivi de vos tickets

https://hub.mesocentre.uca.fr : documentation du Mésocentre

Partie I

- Commandes dans un Système d'Exploitation Linux
  - + rappels
  - + quelques commandes Unix en plus ...
- Environnement de travail
  - + infrastructure (nœuds de calcul, espace de stockage)
  - + outils logiciels en bioinformatique

Commandes Linux pour gérer les droits d'accès

Rappel

u : user, g : group, o : other, a : all
r : read, w : write, x : execute

```
[nadgoue@hpc2 unix]$ ll -h
total 1,3M
-rw-r--r--. 1 nadgoue infra 1,4M Jan 31 16:48 2019formationUnix.pdf
drwxr-xr-x. 2 nadgoue infra    5 Jan 31 15:22 data
-rwxr-xr-x. 1 nadgoue infra  322 Apr 26  2018 mon_prog.sh
-rw-r--r--. 1 nadgoue infra   59 Apr 26  2018 output.txt
```

chmod : changer les permissions (*change mode*)

+ Système logique de notation des permissions

$ chmod [options] [arguments = filename, dirname, ...]

Exemples pour retirer les droits de lecture et d'écriture aux autres :
$ chmod go-rw mon_prog.sh
$ chmod u=rwx,g=,o= mon_prog.sh
$ chmod u=rwx,g=,o= . # rend le répertoire courant (.) traversable

Commandes Linux pour gérer les droits d'accès

- chmod : changer les permissions (*change mode*)

  + Système numérique de notation des permissions : octal (2^3)

1 – can execute		3 (2+1)	: wx
2 – can write	=> Somme :	6 (4+2)	: rw
4 – can read		5 (4+1)	: rx
		7 (4+2+1)	: rwx

Exemple pour retirer les droits de lecture et d'écriture sur un fichier :
$ chmod 700 mon_prog.sh

Commandes Linux pour gérer les droits d'accès

- Tableau récapitulatif des permissions

Binaire	Logique	Décimal
000	---	0
001	--x	1
010	-w-	2
011	-wx	3
100	r--	4
101	r-x	5
110	rw-	6
111	rwx	7

Exemples

$ chmod 600 fichier # l'usager peut lire et écrire le fichier

$ chmod 777 fichier # *all* peuvent lire, écrire et exécuter le fichier

Commandes de redirection Linux

Flux standards

Redirection des flux

2> : redirection des erreurs dans un nouveau fichier plutôt qu'à l'écran

2>&1 : les messages d'erreur sont fusionnés aux données en sortie

Exemple

```
$ sort my_file > result   2> sort.errors
```

Rappel : Environnement et variables d'environnement

- Définition

  Ensemble de règles qui définissent le comportement du système pour un utilisateur. Ces règles sont fixées au moyen de variables d'environnement.

Exemple :

$HOME définit l'emplacement du dossier personnel d'un usager

- Portée des variables d'environnement
  - + le shell courant
  - + la commande env

permet de lister toutes les variables d'environnement du processus shell courant : $HOME, $USER, $SHELL, $PATH, ...

Variables d'environnement

- PATH : contient les chemins d'accès aux répertoires des exécutables

$ echo $PATH
Exemple : /usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin

- HOME : le *home* de l'usager, correspond au *login*

$ echo $HOME # afficher le chemin d'accès absolu au *home* de l'usager
Exemple :
$ /home/nadgoue

- USER : le login utilisateur

$ echo $USER
Exemple :
$ nadgoue

Variables Bash

- Création d'une variable Bash et export dans l'environnement

```
$ env # lister les variables d'environnement
```

Exemples :

```
$ SCRATCHDIR=/storage/scratch/<login>/test
$ echo $SCRATCHDIR
/storage/scratch/<login>/test

$ export TMPDIR=/storage/scratch/$USER/$SLURM_JOB_ID/tmp
$ mkdir -p "$TMPDIR"
$ echo $TMPDIR
/storage/scratch/<login>/<mon_jobid>/tmp
```

L'infrastructure d'un cluster de calcul

Définition

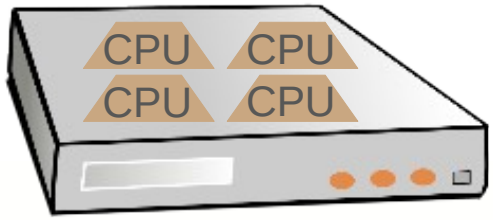
+ Regroupement de plusieurs ordinateurs indépendants situés au même endroit, organisés en grappes

+ chaque machine est un nœud
  faite de processeurs, de mémoire vive, de disques dures,
  de bande passante,
  connectée sur le réseau universitaire,

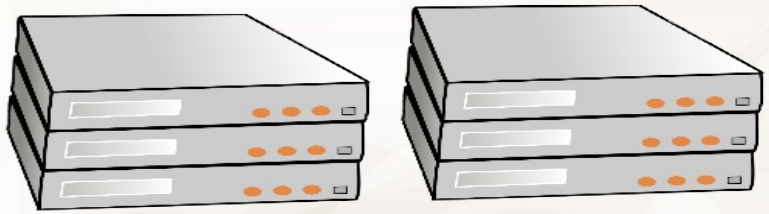
+ configurée en un système homogène (LINUX) :
  un gestionnaire de ressources
  un outil modulant l'environnement
  [protocole de communication entre nœuds]

Vocabulaire pour aborder un cluster de calcul

- Nœud (*Node*) : « ordinateur » avec plusieurs CPU ou GPU, de la mémoire (RAM), de la bande passante

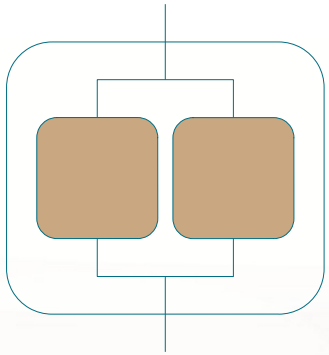


- Partition : groupe de nœuds partageant la même configuration (durée des jobs, ressources, ...)



Vocabulaire pour aborder un cluster de calcul

- CPU : *Central Processing Unit*, Unité physique de calcul
- Cœur logique (*Core*)



1 CPU Dual Core

- Thread : file de traitement par cœur logique

Architecture HPC2

- *High Performance Computing*

  + HPC2

    en production depuis 2015

    ~ 2 500 CPU, 120 To de stockage dédié à la bioinformatique

  + architecture SMP

    Multi-cœurs (384 CPU, 768 cœurs)

    Grande quantité de RAM (11 To)

=> Prochainement HPC3

Les nœuds de calcul sur HPC2

HOME

+ espace 100 Go

+ type stockage

-- données

++ scripts

Frontal :

hpclogin01,

hpclogin02

ssh

Utilisateur

+ Nœuds GPU

piafgpu01, iccfgpu01,

hpcgpu01, ...

+ Nœud hpcsmp01

384 CPU, 12 To RAM

Workers :

+ Nœuds 01 ~ 04

16 CPU, 64 Go RAM

+ Nœuds 05 ~ 08,

hpcphi01

32 CPU, 96 Go RAM

+ Nœuds 09 ~ 24

32 CPU, 128 Go RAM

+ Nœuds 25 ~ 28

32 CPU, 256 Go RAM

+ ...

Le stockage sur HPC2

- Deux types de stockage


/home/<login>


/storage/lab


Utilisateur

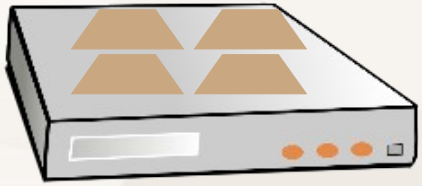
ssh

Frontal

stockage à court, moyen termes


Stockage CEPH

stockage temporaire sur un nœud



Scratch : 500 Go

Libérez la place à la fin des jobs !

/storage/scratch/

Combien ça coûte ??

Prestation	base	**cout UC2A HT**	**cout Ext HT**
HPC2 : 1000h CPU	13,4	17,91 €	22,45 €
SMP : 1000h CPU	38	50,84 €	63,74 €
DGX : 1000h GPU	346,7	463,18 €	580,63 €
OSCAR : 1000h CPU	13,4	17,89 €	22,42 €
CEPH : 1000h To bruts	7,6	10,16 €	12,74 €
S3 / RBD (x3) : 1 To / an		267,13 €	334,87 €

Accès aux outils

Modules d'environnement Lmod

Mécanisme pour gérer dynamiquement les variables d'environnement (PATH, ...)

Intérêts

+ Gérer plusieurs versions d'un outil
+ Gérer les versions par défaut
+ Gérer les dépendances

Commande

$ module

Organisation des modules

Organisation hiérarchique

+ Modules «core» : compilateurs et outils de base
+ Modules «apps» : outils sans dépendance à un compilateur
+ Modules «compiler» : outils dépendants d'un compilateur

Accès aux outils

- Module : gestionnaire de l'environnement des usagers

$ module ou $ module --help # Commande module

$ module list # liste les modules chargés

$ module avail # liste les modules disponibles

$ module whatis # Information sur un module

$ module spider # liste les modules et leurs descriptifs

# Chargement de l'outil dans son environnement
$ module load package1

# Supprimer l'outil de l'environnement
$ module unload package1

# Nettoyage de l'environnement
$ module purge # Attention, purge tous les outils !!!

Accès aux outils

- Modules avec dépendances

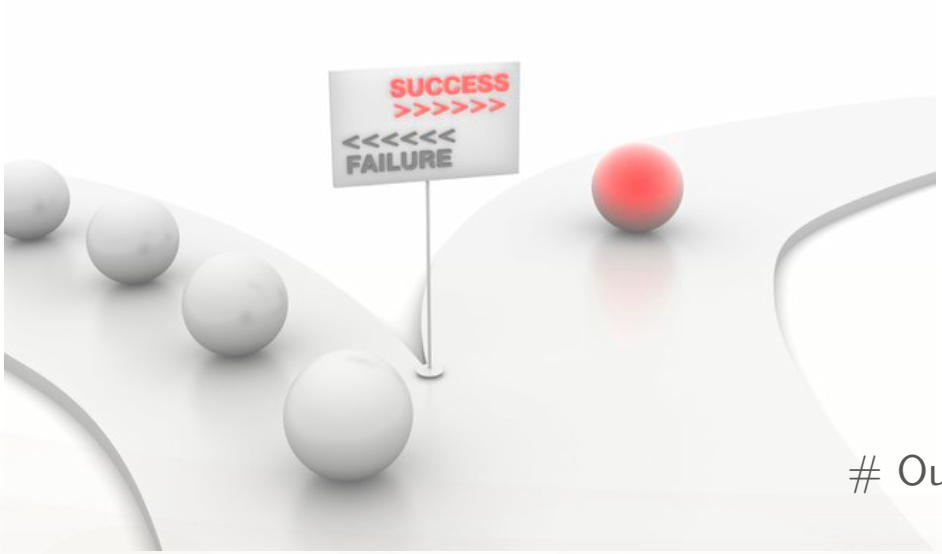
gestionnaire modules →

```
$ module load package1
# Lmod has detected the following error:
Cannot load module "package1".
At least one of these module(s) must be loaded:
package2

$ module load package2 package1
```

Les bonnes pratiques d'installation / utilisation d'outils

sur un poste en local




```
$ sudo apt install monOutil
$ sudo pip install monOutil
```

sur un cluster

```
# Recherche de l'outil
$ module spider monOutil

# Outil présent
$ module load monOutil

# Outil absent mais générique
    # Demande d'installation au SUPPORT

# Outil absent usage unique
$ pip --user install monOutil
```

```
# Outil packagé par conda / bioconda
# Outil disponible dans un conteneur Singularity
```

TP1

- Se connecter au serveur de calcul HPC2

  - à partir d'une invite de commade

$ ssh *<login>*@hpc2.mesocentre.uca.fr

où *<login>* : votre identifiant utilisateur
et hpc2.mesocentre.uca.fr : adresse du serveur

Exemple : syldubo@hpc2.mesocentre.uca.fr

- Répondre aux questions du TP1

Partie II : SLURM

- Commandes de base SLURM
- Notion de jobs
- Contrôle par monitoring

SLURM, pour quoi faire ?

- Définition

  Simple Linux Utility for Ressources Management

- Fonction d'orchestrateur
  - + gestionnaire de ressources (CPU/GPU, RAM, durée)
  - + ordonnanceur de tâches (*jobs*)

Commandes de base SLURM

Commandes	Usage
sbatch	Soumission d'un job
srun	Soumission d'un job interactif Exécution d'une étape dans un job
scancel	Annulation d'un job
sinfo	Informations sur les partitions normal : $<$ 24h ; fast : 4h ; smp : 3j ; long : $<$ 7j
scontrol	Configuration détaillée des partitions
squeue	Informations sur les jobs en attente ou en cours
sstat	Informations sur les ressources consommées par un job en cours
sacct	Informations sur les jobs finis
seff	Job Efficiency Report

Accès aux nœuds de calcul

Mode Interactif – Usage occasionnel

+ pour le déploiement d'un code
+ pour le deboggage

```
$ srun --pty bash
[hpcnodeXX]$
$ srun -p debug --pty bash
[hpcphi01]$ # my fantastic commands ...
```

Mode Batch – Usage principal

+ pour un usage standard
+ accès à un nœud via l'ordonnanceur de jobs (SLURM)

```
$ sbatch scripts/monscript.sh
$ Submitted batch job XXXX
```

Dépendances des jobs

```
$ sbatch --dependency=<liste de dépendances>
```

-- repousse le début du job jusqu'à ce que les dépendances soient satisfaites

```
$ sbatch -d=afterok:jobID monscript.sh
```

Type	Correspondance : le job commence quand ...
after	... le job spécifié a commencé son exécution
afterany	... les jobs spécifiés sont terminés
afterok	... les jobs spécifiés sont terminés avec succès
afternotok	... les jobs spécifiés sont terminés avec échec
singleton	... les jobs spécifiés partageant le même *job-name* et *user* sont terminés

État des ressources

- Files disponibles centrées sur les partitions

```
$ sinfo -l
```

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
normal*	up	1-00:00:00	3	mix	hpcnode[14-15,31]
normal*	up	1-00:00:00	20	alloc	hpcnode[09-13,16-30]
normal*	up	1-00:00:00	13	idle	hpcnode[01-08,32-36]
long	up	7-00:00:00	2	mix	hpcnode[14-15]
long	up	7-00:00:00	10	alloc	hpcnode[09-13,16-20]
long	up	7-00:00:00	2	idle	hpcnode[07-08]
smp	up	3-00:00:00	1	mix	hpcsmp01
debug	up	12:00:00	1	idle	hpcphi01

idle : disponible
mix : partiellement alloué
alloc : complètement alloué

down : inaccessible
drain : vidange des jobs en cours
maint : maintenance

État des ressources

- Files disponibles centrées sur les nœuds

```
$ sinfo -Nle
```

NODELIST	NODES	PARTITION	STATE	...	CPUS	...	MEMORY
hpcnode01	1	normal*	idle		16		63500
...							
hpcnode07	1	long	idle		32		95100
hpcnode07	1	normal*	idle		32		95100
...							
hpcnode09	1	long	allocated		32		128000
hpcnode09	1	normal*	allocated		32		128000

idle : disponible
mix : partiellement alloué
alloc : complètement alloué

down : inaccessible
drain : vidange des jobs en cours
maint : maintenance

État des ressources

$ scontrol show partition normal

```
PartitionName=normal
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=YES QoS=N/A
   DefaultTime=NONE DisableRootJobs=NO
ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=1-00:00:00
MinNodes=1 LLN=NO MaxCPUsPerNode=UNLIMITED
   Nodes=hpcnode[01-36]
   Priority=1 RootOnly=NO ReqResv=NO Shared=NO PreemptMode=OFF
   State=UP TotalCPUs=1088 TotalNodes=36 SelectTypeParameters=N/A
   DefMemPerCPU=2980 MaxMemPerNode=UNLIMITED
```

Exemple : Description d'un job en cours

```
$ scontrol show jobID 22667711
```

```
JobId=22667711 JobName=400mCVA6
   UserId=nadgoue GroupId=infra MCS_label=N/A
   Priority=5238 Nice=0 Account=xxx QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   RunTime=5-17:43:51 TimeLimit=7-00:00:00 TimeMin=N/A
   StartTime=2019-01-10T23:25:42 EndTime=2019-01-17T23:25:42 Deadline=N/A
   Partition=long AllocNode:Sid=hpc2:27897
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=hpcnode18
   NumNodes=1 NumCPUs=32 NumTasks=1 CPUs/Task=32 ReqB:S:C:T=0:0:*:*
   TRES=cpu=32,mem=110G,node=1,billing=32
   MinCPUsNode=32 MinMemoryNode=110G MinTmpDiskNode=0
   OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
   Command=/home/nadgoue/my_prog/my_script.sh
   WorkDir=/home/nadgoue/my_prog
   StdErr=/home/nadgoue/output/slurm-.stderr
   StdIn=/dev/null
   StdOut=/home/nadgoue/output/slurm-22667711.stdout
```

Jobs

Définitions

Job = programme (ex. script Bash) + set d'instructions et de contraintes (allocation CPU, RAM, durée,...)

Script : une succession de commandes

Caractéristiques

+ Soumission d'un job avec la commande sbatch
+ Retourne un identifiant jobID
+ Ressources par défaut :
  - cœurs : 1 CPU
  - mémoire : 3 Go RAM par CPU
  - durée maximale : 24 h
  - partition : normal

Caractéristiques d'un script

- Création d'un fichier, ou fichier existant

$ nano mon_premier_prog.slurm

- Fichier texte

  contenant une succession de commandes

- Contient le pragma en première ligne

  #!/bin/bash

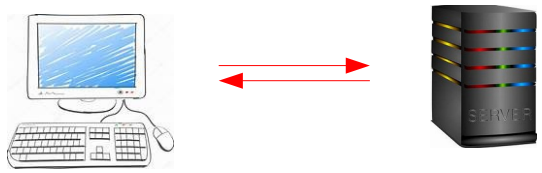
- Fichier exécutable

$ chmod +x mon_premier_prog.slurm

Éditer les scripts

- Plusieurs solutions possibles sur HPC2

  + Une solution (historique) parmi d'autres ... FileZilla



Création d'un hôte hpc2

Choix de l'éditeur

Edit

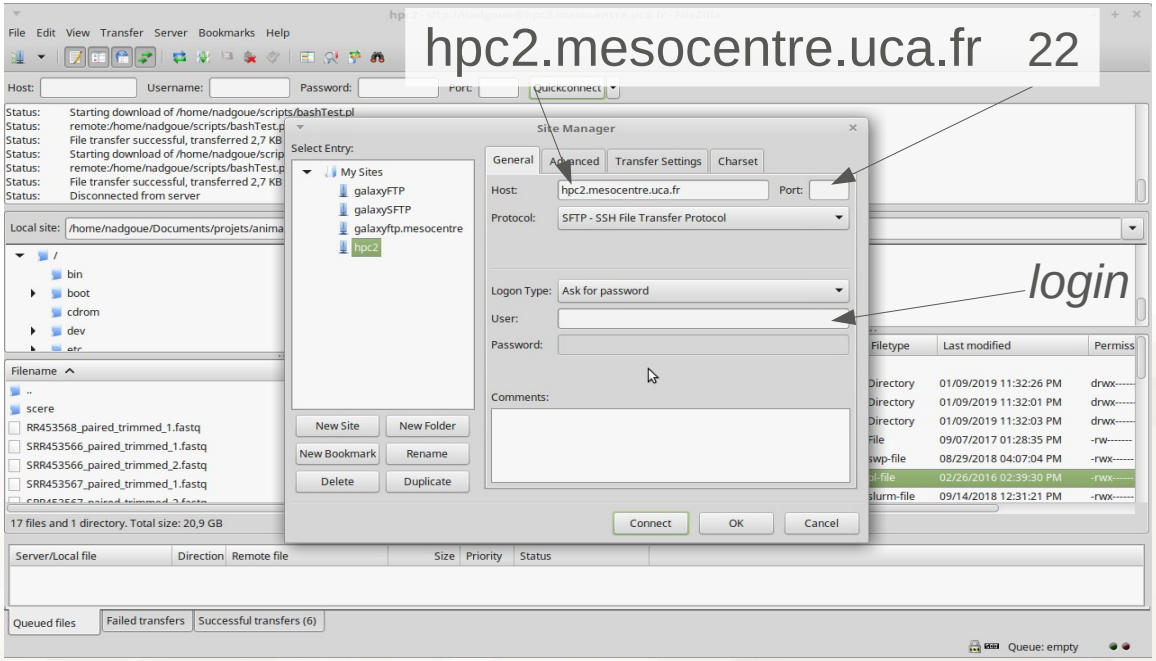
> Settings

> File editing

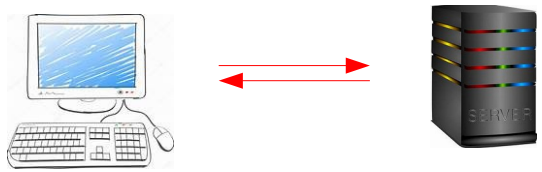
- Use custom editor

exemple /usr/bin/geany

- Always use default editor



Éditer les scripts



- Plusieurs solutions possibles sur HPC2
  - + Une autre solution parmi d'autres ... sshfs

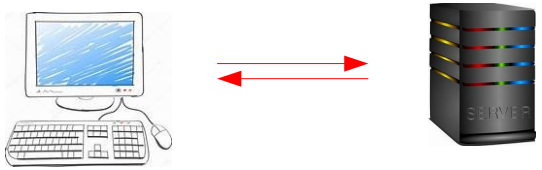
```
# Créer un répertoire temporaire en local
$ mkdir /tmp/my_scripts_on_hpc2
$ cd /tmp

# Monter le répertoire distant
$  sshfs -o idmap=user \
      <login>@hpc2.mesocentre.uca.fr:/home/<login>/scripts ./my_scripts_on_hpc2

# Démonter le répertoire distant
$ fusermount -u ./my_scripts_on_hpc2
```

La suppression du répertoire en local entraîne la suppression du répertoire distant.

Éditer les scripts

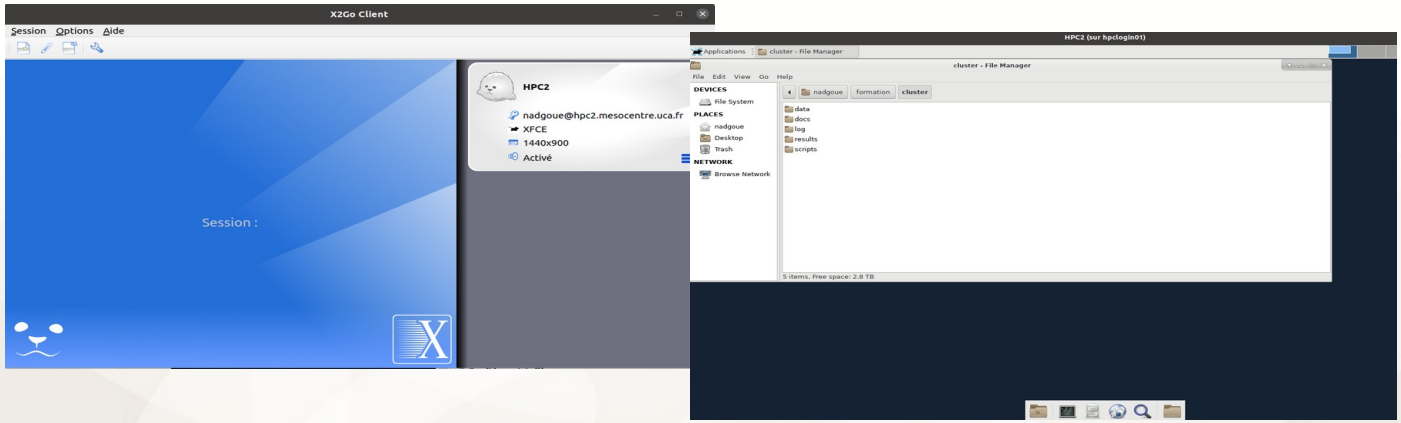


- Plusieurs solutions possibles sur HPC2
  - + Une autre solution parmi d'autres ... X2GO

En local, installer le client X2GO et configurer l'outil

Suivre les recommandations sur

https://hub.mesocentre.uca.fr/docs/cluster/x2go/



+ **Une autre solution très répandue : Visual Studio**

Soumission de jobs

Options de base recommandées

```
-J --job-name      # donner un nom au job
-t --time          # durée d'exécution en jours-heures:minutes
--mem=MB           # quantité de mémoire maximale demandée
--cpus-per-task    # nombre de threads voulus
--temp=MB          # espace nécessaire à l'exécution du job
```

Options de base courantes

```
-n –ntasks         # nombre de tâches / jobs
-o --output=out    # redirection du fichier sortie
-e --error=err     # redirection du fichier contenant des erreurs
```

Soumission de jobs

- En ligne de commande

```
$ sbatch -J premierJob monscript.sh
```

- Dans un script bash
  - + Le répertoire courant du job = répertoire de soumission
  - + La sortie standard et la sortie des erreurs sont redirigées

dans un fichier slurm-JOBID.out

```
#SBATCH --output=slurm-%j.stdout
#SBATCH --error=slurm-%j.stderr
```

  - + Option : Notification par courriel :

```
#SBATCH --mail-user=mon.adresse@uca.fr
#SBATCH --mail-type=ALL
```

Soumission de jobs

- Allocation des ressources : la durée

```
#SBATCH --time=days-hours:min:sec
#SBATCH --partition=<normal,long,smp,fast>
```

Si le job ne se finit pas dans le temps imparti, il est arrêté !!

- Allocation des ressources : la mémoire

```
#SBATCH --mem=<MB>          # par nœud
#SBATCH --mem-per-cpu=<MB>  # par cpu
```

Si le job consomme plus de mémoire que celle réservée,
il est arrêté !!

Utilisation d'un espace de travail temporaire

- Définition du *scratch*
  - \+ *Il est spécifique à chaque nœud (non applicable pour HPC3)*
  - \+ Il est temporaire avec un nettoyage automatique

Mais si vous faites le nettoyage vous-même, c'est mieux !!

  - \+ Il est partagé entre tous les usager d'un nœud

Nouvelle option SLURM
#SBATCH –tmp=50G # exemple d'un job nécessitant 50G de stockage

L'espace /tmp n'est pas approprié pour un usage sur cluster de calcul !

Utilisation d'un espace de travail temporaire

- Description dans le fichier du job

```
#!/bin/bash

SCRATCHDIR=/storage/scratch/$USER/$SLURM_JOB_ID

mkdir -m 600 $SCRATCHDIR
cd $SCRATCHDIR

# code

# rapatriement des résultats dans le répertoire « home » de l'usager
mv $SCRATCHDIR ~/.
```

Monitoring

- Pour les jobs en cours d'exécution
  - squeue
    $ squeue --user=<login>
  - sstat
    $ sstat --format=jobid,AveCPU,AveRSS,AveVMSize,JobID -j *JobID*.batch
  - top ou htop après accès aux nœuds de calcul :

```
# Connexion sur le nœud où tourne le job
[hpc2]$ ssh hpcnodexx
[hpc2]$ srun --nodelist hpcnodexx --pty bash

# Information sur les processus
[hpcnodexx]$ top
ou [hpcnodexx]$ htop
```

Monitoring

- Pour les jobs terminés

$ seff <Jobid>

```
Job ID: 10224794
Cluster: hpc2
User/Group: XXX/XXX
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 6
CPU Utilized: 00:33:52
CPU Efficiency: 37.55% of 01:30:12 core-walltime
Job Wall-clock time: 00:15:02
Memory Utilized (RSS): 2.53 GB
Memory Utilized (VM) : 2.79 GB
Memory Efficiency (RSS): 2.53% of 100.00 GB
```

$ sacct –format=jobid,elapsed,ncpus,state,ReqMem,MaxRSS -j *JobID*

```
sacct -a -o JobID,User,Group,State,Cluster,AllocCPUS,REQMEM,TotalCPU,Elapsed,MaxRSS -j
10224794
```

JobID	User	Group	State	Cluster	AllocCPUS	ReqMem	TotalCPU	Elapsed	MaxRSS
10224794			COMPLETED	hpc2	6	100Gn	33:52.384	00:15:02	
10224794.ba+			COMPLETED	hpc2	6	100Gn	33:52.384	00:15:02	2657364K

~ 2,6 G

CPU * temps écoulé

6 * 00 min 15 sec 02 = 1 min 30 sec 12

Temps total utilisé :

33 min 52 sec !

TP2

- Répondre aux questions du TP2

Calculs distribués

Tableaux de jobs

Job array

Améliorer le temps de calcul

Le calcul est lancé en multi-cœurs

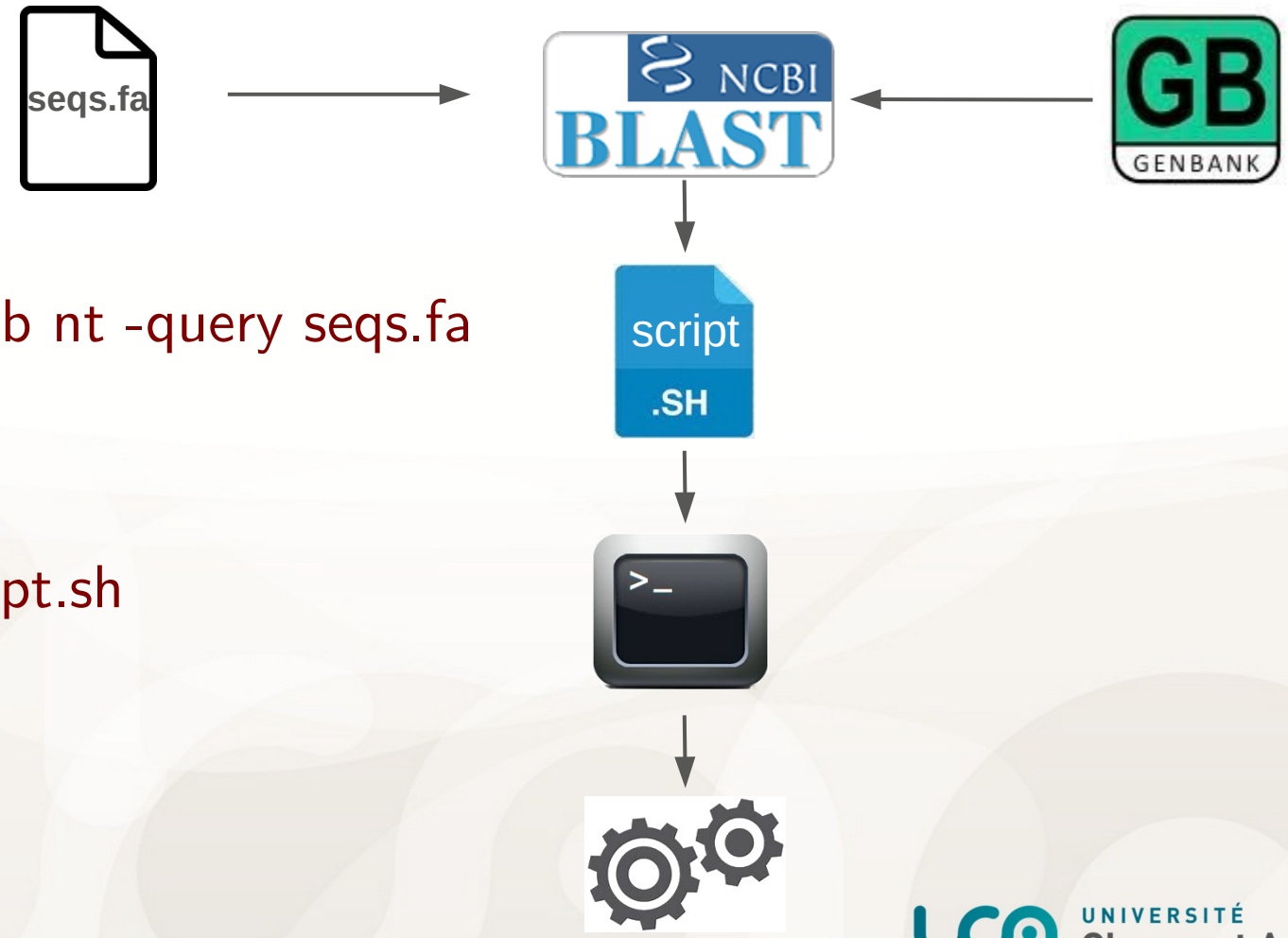
```
#SBATCH --array=<indexes>
```

```
#SBATCH --array=0-9 # création d'un tableau de 10 jobs

tab=(A B C D E F G H I J)
script.py -i ${tab[$SLURM_ARRAY_TASK_ID]} ….
```

Calculs parallélisés : le multi-threading

- Exemple BLAST en Mode basique « mono-CPU »

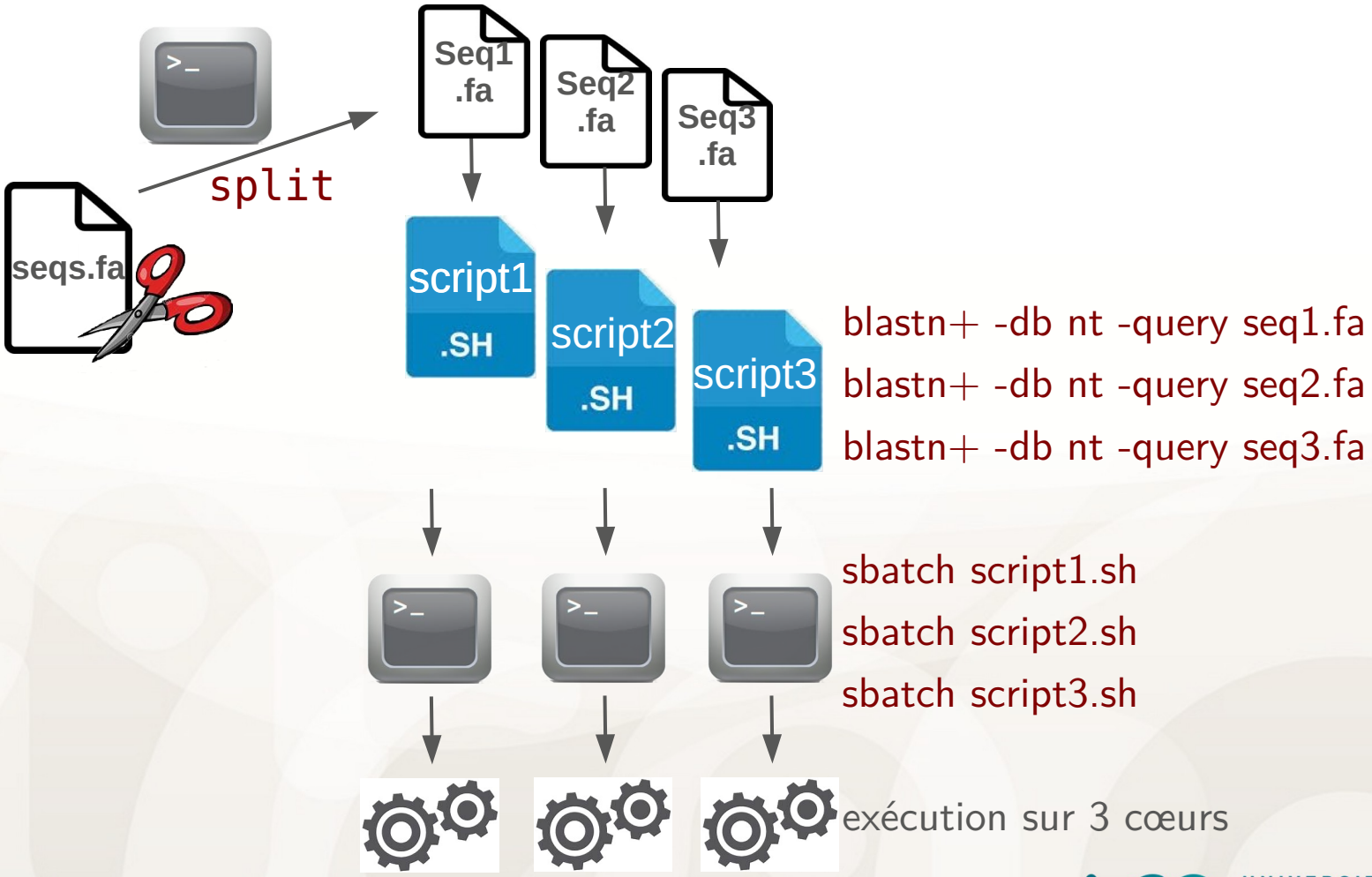


blastn+ -db nt -query seqs.fa

sbatch script.sh

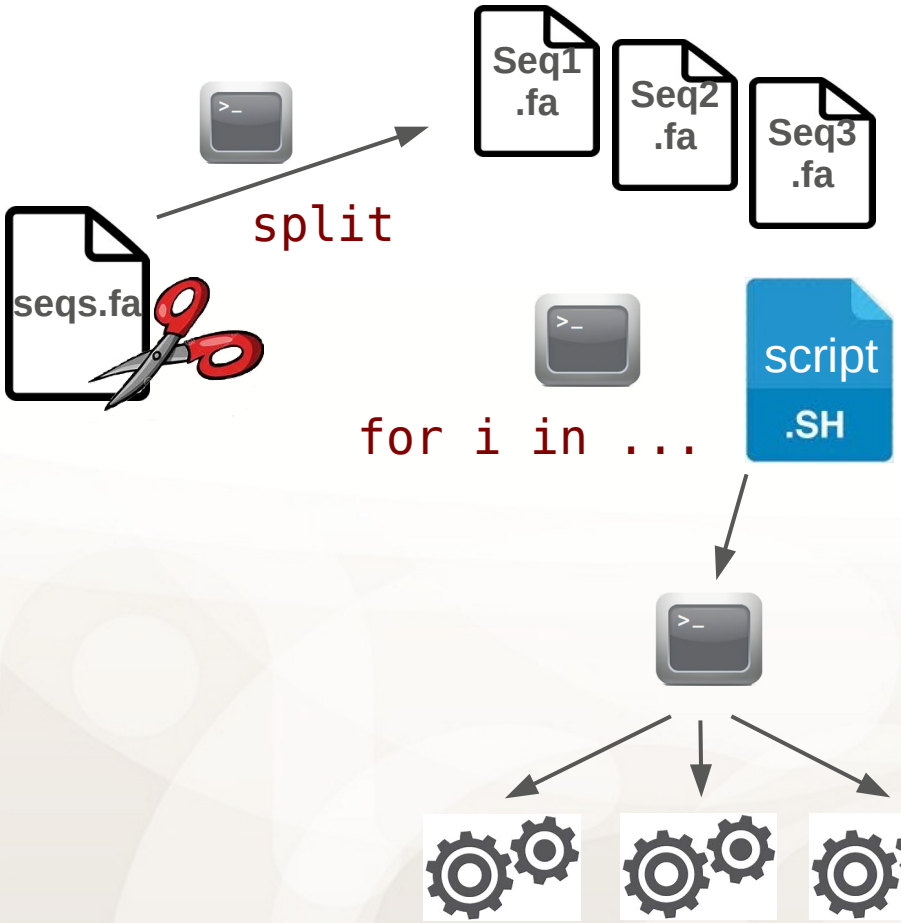
Calculs parallélisés : le multi-threading

- Exemple BLAST en Parallélisation sur les données



Calculs parallélisés : le multi-threading

- Exemple : BLAST en Mode *job array*



sbatch -a=0-2 script.sh

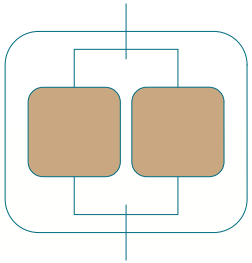
blastn+ -db nt -i seq1.fa
blastn+ -db nt -i seq2.fa
blastn+ -db nt -i seq3.fa

exécution sur 3 cœurs

Calculs parallélisés : le multi-threading

Exemple : BLAST en Mode *Multi-threading*

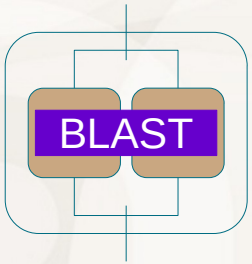
Exemples précédents : 1 job = 2 threads (1 cœur physique)



Si le programme le permet : 1 job = multi-threads

```
blastn+ -num_threads 2 -db nt -i seqs.fa
```

Chaque BLAST utilise 2 cœurs logiques.



Liens utiles

- Guide Lmod et modules

http://lmod.readthedocs.io/en/latest/010_user.html

- SLURM

https://slurm.schedmd.com/sbatch.html
https://slurm.schedmd.com/tutorials.html

- Aide au DÉBOGGAGE

https://stackoverflow.com

TP3

- Répondre aux questions du TP3