

Outline

1. Motivation
2. Presentation of the DCSR
3. General High Performance Computing concepts
4. UNIL HPC cluster infrastructure overview
5. Cluster storage locations
6. Slurm commands

Storage locations

Storage locations overview



Home directory

Backed-up storage for private scripts, small data. Limited to 5 GB of space.



/scratch/axiom and /scratch/wally

Storage for large input / output files you are currently working on. No back-up !



UNIL NAS (Isilon)

Medium to long-term storage, with fast access. Not always backed-up (project dependent)



Archive

Backed-up long term storage, with slow access.

Home directory: a user's private space



Features:

- **Backed-up** and **versioned** storage.
- Limited to **5 GB** of space.
- `/users/<user name>`
- Default location after logging-in to the cluster.

Use case:

- Ideal for storing your **scripts** and other important **small files**.

/scratch directories: per-project shared workspace



Features:

- Large storage space.
- No back-up.
- Storage price: 95 CHF TB/year.
- For each new project a directory is automatically created in the directory of the PI in charge of the project.

/scratch/axiom/FAC/<Faculty>/<Department>/<PI user name>/<project>

- Each PI has a “default” project.

/scratch/axiom/FAC/<Faculty>/<Department>/<PI user name>/default

Use case:

- Storage of input / output files of the projects you are currently working on.

/scratch directories: per-project shared work space



/scratch/axiom vs. /scratch/wally

- **Same directory structure.** When a project is created, a /scratch directory for the project is created on both /scratch/axiom and /scratch/wally.
- **No synchronization** between axiom and wally /scratch.
- /scratch/axiom only accessible from axiom cluster nodes, and /scratch/wally only accessible from wally nodes.
- Both /scratch locations accessible from the front-end machines.

NAS (Isilon): medium to long-term storage



Features:

- **Large storage space** with a per-project quota (ask your PI for details).
- Depending on the project, can be backed-up or not.
- **Storage price:** 48 CHF (no backup) or 96 CHF (with backup) per TB/year.
- Data stored on disk => faster access than **Archive**.
- Accessible **only from font-end machines** via the movedat command.

Use case:

- Ideal for keeping data over a project's lifetime where frequent access is needed.

Archive: long-term storage



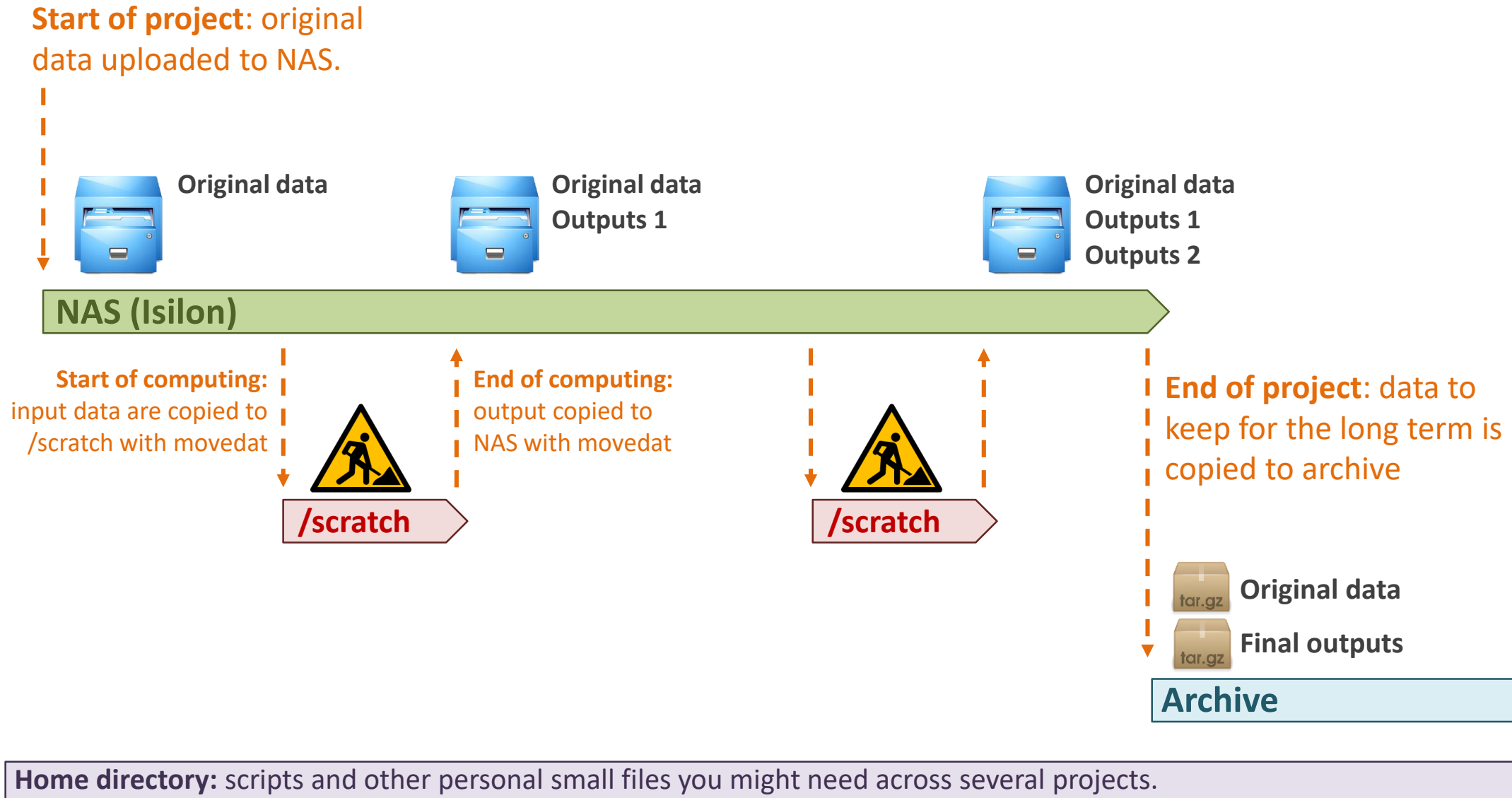
Features:

- “**unlimited**” storage **space** (currently > 0.5 PB).
- **Backed-up** and free.
- Magnetic tape based storage = **slow** read/write **access**.
- Accessible **only from font-end machines** via the movedat command.

Use case:

- Ideal for archiving data that you do not access frequently over longer time periods.
- Avoid archiving large numbers of small files. Group files together instead, e.g. by creating a .tar.gz archive.

Data flow summary: where to keep data during your project



movedat command - access data from **NAS** and **Archive**



General syntax:

List content of directory on NAS:

```
movedat <user name>@dexwin.unil.ch:/path/to/directory
```

Copy data from NAS to /scratch:

```
movedat <user name>@dexwin.unil.ch:/path/to/file/to/copy /scratch/axiom/custom/path
```

Copy data /scratch to NAS:

```
movedat <file to copy> <user name>@dexwin.unil.ch:/path/to/destination/directory
```

Notes:

- to access Archive, use “dexvital.unil.ch” in the commands above.
- currently only former Vital-IT users have access to Archive.
- When copying large amounts of data, it’s best to run the movedat behind a “nohup” command, so that the transfer continues even if your session is ended.

movedat command - examples



List content of directory on NAS:

```
[alice@login1 ~]$ movedat alice@dexwin.unil.ch:CTR/CI/DCSR/cours_intro_hpc/alice/
Password for alice@dexwin.unil.ch:
drwxrwxr-x          Aug 17 2010 09:30  test_dir/
-rw-rw-r--      1,925,637  Aug 17 2010 20:07  sample_1.txt
-rw-rw-r--      1,925,637  Aug 17 2010 16:48  sample_2.txt
-rw-rw-r--      1,925,637  Aug 17 2010 15:08  sample_5.txt
```

Copy data from NAS to /scratch:

```
[alice@login1 ~]$ nohup movedat alice@dexwin.unil.ch:CTR/CI/DCSR/cours_intro_hpc/alice
                    /scratch/wally/CTR/CI/DCSR/cours_intro_hpc/alice/ &
38.8 megabytes      9937 files at   16.5 megabits/sec
dexwin.unil.ch:CTR/CI/DCSR/hhussain/cours_intro_hpc/D2c/10Kfiles_rand_4K  39.1 MB (10000
files) in 20.6 sec (15.9 mbit/s)
```

The movedat command is run with nohup.

movedat caches the authentication credentials,
so the password is not asked again.

Outline

1. Motivation
2. Presentation of the DCSR
3. General High Performance Computing concepts
4. UNIL HPC cluster infrastructure overview
5. Cluster storage locations
6. Slurm commands

Running jobs on the cluster:

sbatch and *srun*

Submit jobs to the cluster: *sbatch* and *srun*

- 3 different possibilities to submit jobs to the cluster:

Most frequent case

sbatch

submit a script.

```
sbatch <script.sh>
```

```
#!/bin/bash
# Slurm options:
#SBATCH ...
#SBATCH ...
#SBATCH ...
# Run commands:
...
```

srun

submit a single command.

```
srun <options> <cmd to run>
```

sbatch + *srun*

submit a script to *sbatch* that contains *srun* commands.

```
sbatch <script.sh>
```

```
#!/bin/bash
# Slurm options:
#SBATCH ...
# Run tasks:
srun <options> command
srun <options> command
...
```

sbatch vs. *srun*: when to use what

sbatch: for regular job script submission.

- parameters can be passed in script with the **#SBATCH** keyword. This improves reproducibility as commands + options are in the same file.
- standard output/error streams are always saved to a file.

```
#!/bin/bash
# Slurm options:
#SBATCH ...
#SBATCH ...
#SBATCH ...
# Run commands:
...
```

srun: for interactive jobs or small single command tests.

- standard output/error streams are shown directly in the user's shell.
- job runs in the foreground (unless "&" is passed).
- **Warning**: #SBATCH instructions in scripts are not recognized.
- **Warning**: srun only accepts to run scripts that have execution permission.

sbatch + **srun**: for special/advanced use cases.

- If accounting at the job task level (i.e. job steps) is needed.
- Jobs with parallel tasks requiring specific resource allocation (e.g. MPI jobs.)

```
#!/bin/bash
# Slurm options:
#SBATCH ...
# Run tasks:
srun <options> command
srun <options> command
```

srun

run interactive jobs and tests

srun – running a single command

General syntax: `srun <options> command to run`

With *srun*, stdout/stderr are shown directly in the terminal.

Examples:

```
[user@login1 ~]$ srun hostname  
cpt005.wally.unil.ch
```

```
[user@login1 ~]$ srun -p normal --time 01:00:00 --cpus-per-task 2 --mem 4GB  
singularity run /software/singularity/containers/R-3.1.1-1.centos7.simg  
#####  
### List of packages available R 3.1.1:  
### *****  
[1] "acepack"          "annotate"        "AnnotationDbi"   "base"  
[5] "base64enc"        "BatchJobs"       "BBmisc"         "Biobase"  
...  
[93] "tools"           "utils"           "XML"            "xtable"  
[97] "XVector"  
#####  
[user@login1 ~]$
```

Things to keep in mind with *srun*

- *srun* will ignore any #SBATCH options passed in a script, considering them simply as bash comments. Therefore it's best to use *srun* only for single commands (tests) and use *sbatch* when submitting scripts.
- with *srun*, if you disconnect from the front-end machine or otherwise kill your terminal, the job will also get killed (whereas a job submitted with *sbatch* will continue to run). **

** unless you run your srun inside a “screen” or “nohup” command.

srun – interactive jobs

- Passing the option **--pty bash** tells ***srun*** to start an interactive job.
- Useful to test commands / debug your scripts (instead of running rests directly on the front-end machine).
- Type “**exit**” or “**Ctrl + D**” to exit the interactive job.

General syntax: **srun <options> --pty bash**

Example:

```
[user@login1 ~]$ hostname
login1.wally.unil.ch
[user@login1~ ]$ srun -p normal -t 1:00:00 --cpus-per-task=2 --mem=4GB --pty bash
[user@cpt002 ~]$ hostname
cpt002.wally.unil.ch
[user@cpt002 ~]$ which samtools
/usr/bin/which: no samtools
[user@cpt002 ~]$
[user@cpt002 ~]$ module load Bioinformatics/Software/vital-it
[user@cpt002 ~]$ module add /software/module/UHTS/Analysis/samtools/1.8
[user@cpt002 ~]$ which samtools
/software/UHTS/Analysis/samtools/1.8/bin/samtools
[user@cpt002 ~]$
[user@cpt002 ~]$ exit
exit
```

sbatch


submit job scripts

sbatch – submit job scripts

sbatch is the recommended way to submit regular jobs to the cluster.

General syntax:

`sbatch jobScript.sh`
`sbatch ./jobScript.sh`



```
#!/bin/bash

# Slurm options:
#SBATCH <option>
#SBATCH <option>
#SBATCH <option>

# Run commands:
...
```

- Best practice is to put all your options and commands into a single script.
- Options in script must be prefixed with **#SBATCH**.

Example:

```
[user@login1 ~]$ sbatch ./variant_calling.sh
Submitted batch job 28873
[user@login1 ~]$
[user@login1 ~]$ squeue -u $USER
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
28873	normal	testJob	user	PD	0:10	1	cpt007

sbatch – job scripts structure

1. Interpreter line.

- Specifies the shell that will be interpreting the commands in the script.
- This line is **compulsory**: all job scripts must start with: **#!/bin/bash**

2. Option sections.

- One option per line.
- All options lines must start with **#SBATCH**.

3. Commands to run, written in bash.

- A number of slurm environment variables can be used (e.g. \$SLURM_JOB_ID).

```
#!/bin/bash

# Slurm options
#SBATCH --account=HPC_test
#SBATCH --partition=normal
#SBATCH --time=02:30:00
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=32G
#SBATCH --job-name=rJob
#SBATCH --export=NONE

# Commands start here
module add R/351
cd /scratch/wally/custom/path
mkdir run_${SLURM_JOB_ID}
cd run_${SLURM_JOB_ID}
Rscript doSomething.R
...
```

sbatch / srun options

Long and short option names in slurm

As you will see, many of the slurm options have a corresponding “single dash + single letter” abbreviated form:

For instance:

--account	→	-A
--partition	→	-p
--output	→	-o
--error	→	-e
--job-name	→	-J



- Using the long option, the separator between the option and its value can be either an “=” or a space “ ”.
- Using the short option, the separator can only be a space “ ”.

```
#!/bin/bash
```

```
#SBATCH --partition=long ✓
```

```
#SBATCH --partition long ✓
```

```
#SBATCH -p long ✓
```

```
#SBATCH -p=long ✗
```


Number of nodes and tasks assigned to a slurm job.

Reminder: tasks are sub-allocations that run in parallel. If you have several sequential tasks requesting **--ntasks=1** is sufficient.



long option	short	description
--nodes	-N	Number of nodes requested for the global allocation.
--ntasks	-n	number of tasks across all requested nodes. Note that unless you specify the --ntasks-per-node option, tasks are distributed on requested nodes based on resource availability.
--ntasks-per-node		Specifically assign n tasks per node rather than having slurm assign them based solely on resource availability.

--ntasks and **--ntasks-per-node** are mutually exclusive.



Important: if all tasks should be executed on the same node, you should specify **--nodes=1**, otherwise slurm will likely distribute tasks across different nodes (depending on resource availability).

Most frequent case

Examples:

Single task job (most cases).

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
```

10 tasks, “randomly” distributed.

```
#!/bin/bash
#SBATCH --nodes=5
#SBATCH --ntasks=10
```

10 tasks, but exactly 2 per node.

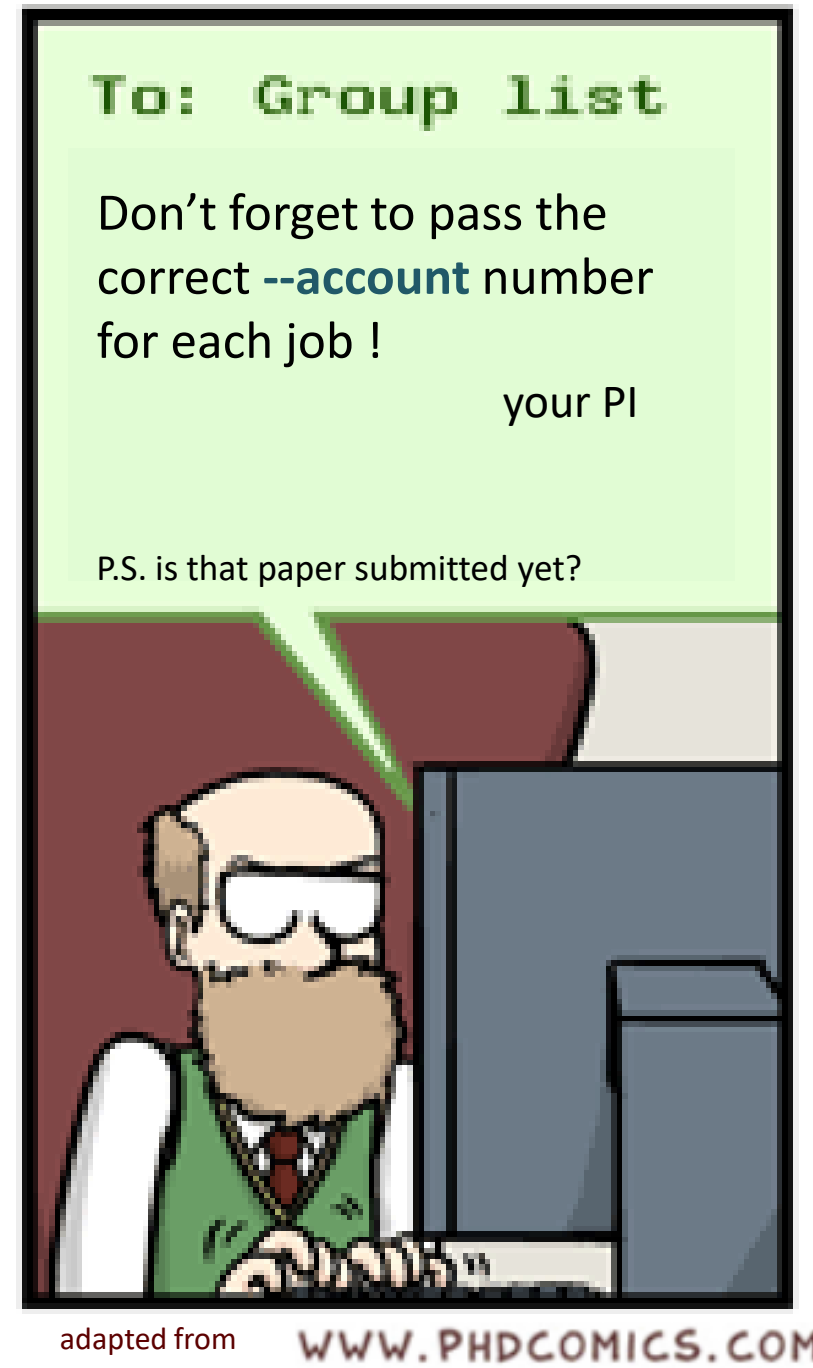
```
#!/bin/bash
#SBATCH --nodes=5
#SBATCH --ntasks-per-node=2
```

--account: resource usage tracking for billing!

long option	short	description
--account	-A	Charge resources used by this job to the specified account.

- ➔ The **--account** option is needed so that your resource usage can be billed correctly to your PI.
- ➔ It has no effect on your actual computing.
- ➔ If **--account** is omitted, your job will be charged to your default account.

A message from your PI →



Which slurm accounts are associated with me ?

Use the following command to list the **--account** values associated your user name:

```
[bob@login1~]$ sacctmgr show user $USER WithAssoc format=User,Account%30,DefaultAccount%30
```

User	Account	Def Acct
bob	cours_intro_hpc	core_it
bob	core_it	core_it

User name

List of all accounts the user is associated with.

Default account (this will be used if you do not specify **--account** when submitting a job.

Examples:

```
[bob@login1 ~]$ srun --account=cours_intro_hpc --time=2:00:00 --pty bash
```

```
[bob@login1 ~]$ sbatch --account=core_it jobScript.sh
```

```
[bob@login1 ~]$ srun --pty bash
```

← If no account is specified, the job's resource usage are charged to the default account.



Select a partition, name your job

long option	short	description
--partition	-p	Specify partition (i.e. queue). Currently 4 partitions are available: normal, long, ax-normal, ax-long.
--job-name	-J	Give a specific name to your job.

➔ Select the correct **--partition** depending on your job's execution time:

- **normal** for jobs < 24h.
- **long** for jobs up to 10 days.
- The default partition is **normal**.

➔ **--job-name** is not essential but can be convenient.

Examples:

Long options:

```
#!/bin/bash

#SBATCH --account=PIname_project
#SBATCH --partition=normal
#SBATCH --job-name=testRun

# Commands start here...
```

"normal" is the default, but for reproducibility it's always good to specify it explicitly.


Short options:

```
#!/bin/bash

#SBATCH -A PIname_project
#SBATCH -p long
#SBATCH -J testRun

# Commands start here...
```

Reminder:

With slurm short options, a space must be used instead of "=". 

Save standard output/error to file

- The **--output** and **--error** options are used to specify files where to save stdout and stderr streams.
- If **--output** is specified but **--error** is not, both stdout/stderr are written to the same file.

long option	short	description
--output	-o	Write the job's standard output directly to the specified file.
--error	-e	Write the job's standard error directly to the specified file.



Warning: if an output file already exists, slurm will overwrite it!

- Slurm **always** saves the stdout/stderr, even if the **--output** or **--error** options are not passed. By default both are saved to a same file, named **slurm-<jobID>.out**, in the current working directory.
- Slurm writes to stdout/stderr files in real time (i.e. as the job progresses). To watch these files live, “tail -f <filename>” can be used.

Filename patterns

To make the naming of stdout/stderr output files easier, slurm provides a number of variables that expand at runtime. The most useful are listed here:

variable	expands to
%j	job ID. Very useful to ensure the stdout/stderr files have unique names.
%x	job name which is passed using the --job-name option.
%u	user name.
%N	name of host running the job.
%J	jobID.stepID of the current task (only applies if tasks submitted with srun).



Filename patterns only work in filenames, and only in the slurm options:

- Things like **--job-name=testJob_%j** does not work (%j will not expand).
- They will not work in your bash commands after the option section.

Examples:

```
#!/bin/bash
#SBATCH --job-name=testRun
#SBATCH --output=%x_%j.out
#SBATCH --error=%x_%j.err
```

Assuming this job gets ID 1859, the stdout/stderr output files will be:

- testRun_1859.out
- testRun_1859.err

```
#!/bin/bash
#SBATCH -o logDir/random_%j.out
#SBATCH -e logDir/random_%j.err
```

Output files can also be saved to a different location than the current working directory.

--time option: set a job's run time limit

- By default, a job's maximum runtime is set to the partition's default value: 12h for normal/ax-normal, 5 days for long/ax-long.
- This can be changed with the **--time** option:

long option	short	description
--time	-t	Set limit to the run time of a job (wall time). If the runtime limit is reached, the job gets killed. Accepted time formats: <ul style="list-style-type: none">• hours:minutes:seconds• days-hours:minutes:seconds• days-hours:minutes• minutes:seconds• minutes



Warning: if the requested time limit exceeds the partition's time limit, the job will be left in a PENDING state... **indefinitely!**

--time option: examples

Acceptable time format examples for the --time option.

```
#!/bin/bash
#SBATCH --partition=normal
#SBATCH --job-name=testRun

#SBATCH --time 2:30:00      # 2 h 30 min.
#SBATCH --time 0-2:30      # 2 h 30 min.
#SBATCH --time 2:30        # 2 min. 30 sec.
#SBATCH --time 3-00:00:00   # 3 days.
```

Note the difference between 2h30 and 2 min 30 sec. !



Problem when time limit > max runtime of partition.

Note: The --time parameter is passed outside of the script. This is not recommended and is only done here so you can see the passed value.

```
[user@login1 ~]$ sbatch --time 3-1:00:00 basic_cpuStress.sh
Submitted batch job 1433
[user@login1 ~]$
[user@login1 ~]$ squeue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
1433	normal	test	user	PD	0:00	1	(PartitionTimeLimit)

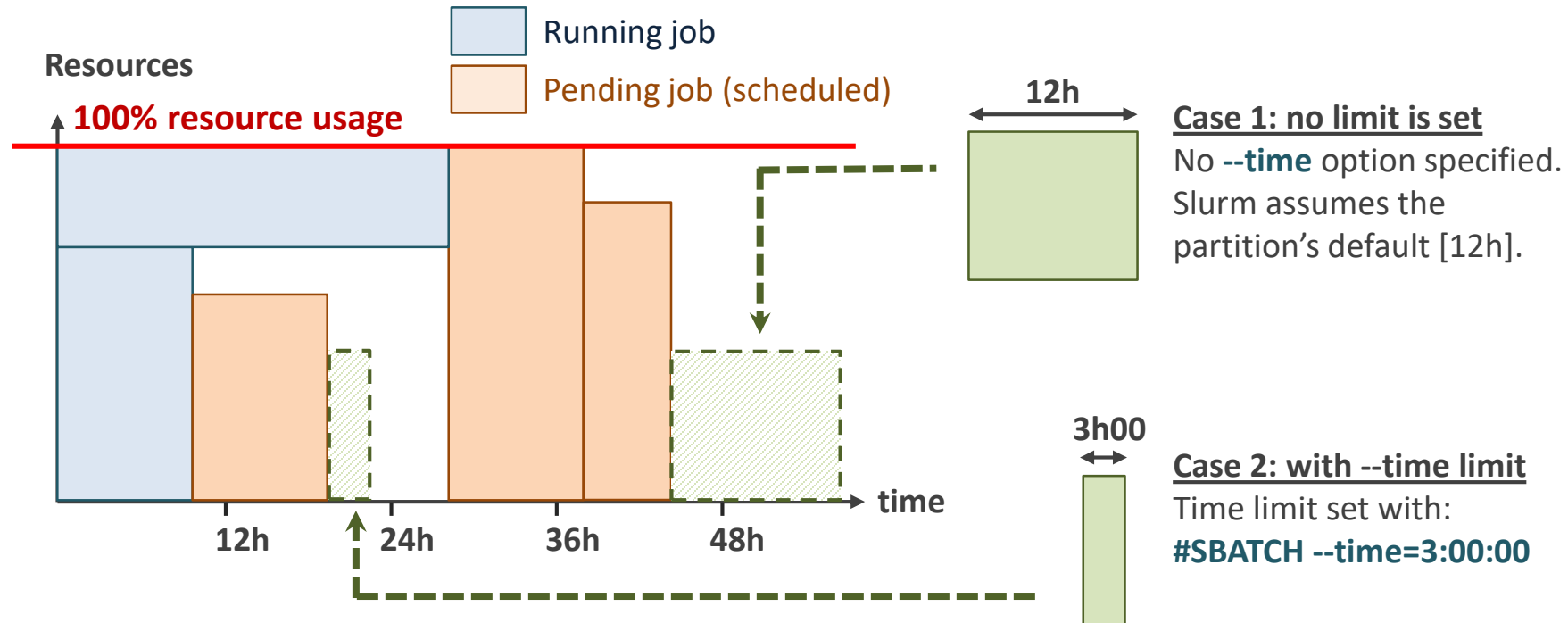
Normal partition has a max runtime of 24h

PD = PENDING

Hint that something isn't quite right !

Why use `--time` to set a restrictive time limit to my job when queues are time limited anyway ?

- ➔ Let's assume we want to run a job that will last about 2 hours. Since it's less than 24h, we will chose to submit it to the “normal” partition.



Altruism meets personal fitness:

- Specifying `--time` allows slurm to better optimize the usage of the cluster.
- Benefits your jobs directly as slurm can “squeeze-in” jobs in free slots smaller than the queue’s default time limit.
- Particularly interesting if the job is much shorter than the default limit of the queue.

Stay tuned: slurm email notifications

Slurm can send notification emails to help monitoring your jobs.

long option (no short option available)	description
--mail-user <email address>	Email address to send notifications to.
--mail-type <type>	Type of events for which SLURM should send an email. Default value is ALL. Events types (non-exhaustive list) <ul style="list-style-type: none">BEGIN, END, FAIL.TIME_LIMIT_90 (reached 90 percent of runtime limit).TIME_LIMIT_80 (reached 80 percent of runtime limit).TIME_LIMIT_50 (reached 50 percent of runtime limit).ALL (equivalent to: BEGIN,END,FAIL).NONE (no notification).

Example:

```
#!/bin/bash
#SBATCH --mail-user alice@sib.swiss
#SBATCH --mail-type BEGIN,END,FAIL,TIME_LIMIT_80
```

The emails only contain a subject line – the body itself is empty.



	Subject	Correspondents
success	Slurm Job_id=1362 Name=test Ended, Run time 00:00:30, COMPLETED, ExitCode 0	Slurm batch scheduler user
start	Slurm Job_id=1362 Name=test Began, Queued time 00:00:01	Slurm batch scheduler user
failed job	Slurm Job_id=28924 Name=sleepJob Failed, Run time 00:00:20, FAILED, ExitCode 1	slurm scheduler user
memory exceeded	Slurm Job_id=28920 Name=memBurn Failed, Run time 00:00:59, OUT_OF_MEMORY	slurm scheduler user

Passing options to *sbatch* outside of script

- Job options can be passed directly to *sbatch*, outside of a script.
- Options passed directly to *sbatch* take precedence over options passed inside a script.

```
[user@login1 ~]$  
[user@login1 ~]$ sbatch --partition=long --mem=48G exploreJob.sh  
Submitted batch job 1492  
[user@login1 ~]$
```

A word of caution...



passing options directly to slurm is OK for a quick test, but don't make a habit of it.

not recommended because **not reproducible** !

What could go wrong ?

- yourself running the script after 6 months.
- a colleague to whom you passed your script.

slurm environment variables

- Slurm environment variables provide access to useful values that can be used in scripts submitted to *sbatch*.
- Here is a (non-exhaustive) list of the most useful of them:

variable name	description
\$_SLURM_JOB_ID	Job ID value.
\$_SLURM_JOB_NAME	Job name – i.e. value passed to --job-name.
\$_SLURM_CPUS_ON_NODE	Number of CPUs allocated on node running the job.
\$_SLURM_CPUS_PER_TASK	Number of CPUs allocated to current task.
\$_SLURM_SUBMIT_DIR	The directory from which sbatch was invoked or, if applicable, the directory specified by the --chdir option.
\$_SLURM_JOB_NODELIST	List of nodes allocated to the job.



In array jobs, some environmental variables change:

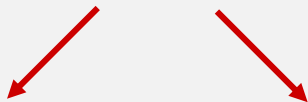
variable name	description
\$_SLURM_ARRAY_JOB_ID	Job array master ID value. In array jobs, this must be used instead of \$_SLURM_JOB_ID.
\$_SLURM_ARRAY_TASK_ID	Job array index ID number of the current array replicate.

Why and how to use environment variables in your scripts ?

```
#!/bin/bash
```

```
#SBATCH --account=324517
#SBATCH --partition=normal
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --job-name=testJob
#SBATCH --export=NONE
```

Print info to stdout to keep a record of your job name or ID number.



```
# Commands start here:
```

```
echo "Starting job $SLURM_JOB_NAME with ID $SLURM_JOB_ID".
```

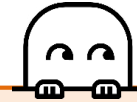
```
outputDir=run_$SLURM_JOB_ID
mkdir $outputDir
```

← Create unique output directories that are unique to each job (avoid overwriting outputs)

```
samtools sort --threads $SLURM_CPUS_PER_TASK input.sam > $outputDir/sorted.bam
```

← Match the number of threads in your software to the number of CPUs in your allocation (--cpus-per-task option).

Warning: *sbatch* exports your local environment !



slurm exports (by default) **the local shell environment** (from the front end machine) **to the compute nodes that runs the job.**

To avoid this, it is best to always add the following option to your script:

```
#SBATCH --export=NONE
```

This will make sure your environment is clean at the start of your job.



Even with **--export=NONE**, your bash profile (“~/.bashrc” file) will still be loaded when your job starts. So please keep it clean :-)



Example without --export=NONE.

```
[user@login1 ~]$ export THIS_IS_SECRET="Don't tell anyone"
[user@login1 ~]$ module add /software/module/UHTS/Analysis/samtools/1.8
[user@login1 ~]$ sbatch testScript.sh
...
[user@cpt05 ~]$ module list
Currently Loaded Modulefiles:
  1) UHTS/Analysis/samtools/1.8
[user@cpt05 ~]$ echo $THIS_IS_SECRET
Don't tell anyone
[user@cpt05 ~]$
[user@cpt05 ~]$ module purge
[user@cpt05 ~]$ module list
No Modulefiles Currently Loaded.
[user@cpt05 ~]$
```

This happens inside "testScript.sh"...

Example with #SBATCH --export=NONE.

```
[user@login1 ~]$ export THIS_IS_SECRET="Don't tell anyone"
[user@login1 ~]$ module add /software/module/UHTS/Analysis/samtools/1.8
[user@login1 ~]$ sbatch testScript.sh
...
[user@cpt05 ~]$ module list
No Modulefiles Currently Loaded.
[user@cpt05 ~]$ echo $THIS_IS_SECRET

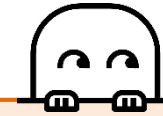
[user@cpt05 ~]$
```

This happens inside "testScript.sh"...

exercise 4 + 5

slurm: requesting resources

A word of caution...



The higher the resource requirements, the more difficult to find an available compute node.

Do not request (much) more than you need

- more time for your job to get started.
- Unused resources are unavailable to others, as well as for your own other jobs.

Requesting multiple CPUs

- By default slurm allocates **1 CPU per task** (and 1 task per job).
- Additional CPUs must be explicitly requested with the **--cpus-per-task** option.

long option (no short option available)	description
--cpus-per-task	Number of processors to be allocated to each task. <ul style="list-style-type: none">• All processors for a task are allocated on a same node.• Without this option slurm will allocate only one processor per task.

Warning



slurm strictly enforces the CPU limit allocated to each job. If you ask for 1 processor – or don't specifically ask for more – your job will have only 1 processor available, and if you then run something multi-threaded all threads will share this one processor. So if more than 1 processor is needed, it is essential to explicitly request them.

Requesting multiple CPUs: examples

General case example:

Job is run as a single task on a single node.

For instance:

- shared memory (e.g multi-threaded).
- jobs arrays.

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=testJob
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=24

# Commands start here ...
cd /path/to/workdir
module add UHTS/Analysis/...
```

Total allocation: 24 CPUs on a single node.

Advanced case example:

Job runs multiple tasks, e.g. “MPI” jobs.

The total number of CPUs for the job allocation is:

$$\text{<number of tasks> * cpus-per-task}$$

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=mpiJob
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=3
#SBATCH --cpus-per-task=8

# Commands start here ...
srun ...
srun ...
srun ...
```

Total allocation: 24 CPUs on a single node.

Each task (started with **srun**) can access only 8 CPUs.



Requesting memory

- ➔ By default, jobs get allocated 2 GB of memory per CPU (on all partitions).
- ➔ Jobs that exceed their memory limit get killed by slurm.
- ➔ To request more memory, two **mutually exclusive** options that can be used:

long option (no short option available)	description
<code>--mem=<memory>[M G T]**</code>	memory limit per node .
<code>--mem-per-cpu=<memory>[M G T]**</code>	memory limit per cpu (logical processor).

** 'M','G' or 'T' after the memory value specifies the unit: megabyte [M], gigabyte [G] or terabyte [T].
If no unit is given, slurm defaults to megabytes [M].

For jobs running on a single node, both `--mem` and `--mem-per-cpu` can achieve the same result. For MPI jobs distributed over several nodes, only `--mem-per-cpu` should be used.



Example: 2 equivalent ways of requesting 3 CPUs and 24 GB of memory on a single compute node.

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=3
#SBATCH --mem=24000
#SBATCH --mem=24000M
#SBATCH --mem=24G
```

} These are equivalent.
Select one.

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=3
#SBATCH --mem-per-cpu=8G
```

How do I know how much resources I need ?

- CPU:**
- Read the software's documentation: hopefully it will indicate whether it is multithreaded or not.
 - (good) multithreaded software will have an option allowing you to select how many CPUs the software should use (e.g. samtools has a --threads option).
 - Look at the output of "*sacct*" and compare values in columns "CPUTime" and "TotalCPU".

- Memory:**
- Make a first run where you set --mem to your best "guesstimate"...
 - If job is terminated with "OUT_OF_MEMORY" error, increase memory allocation and try again...
 - If job completes, use "*sacct*" and compare the "MaxRSS" column to your --mem request. If you requested much more than the actual usage, decrease your memory request.

Example:

```
[user@login1 ~]$ sacct --format=jobid,jobname,partition,account%30,alloctres%40,cputime,
totalcpu,maxrss,submit,state%15,exitcode,elapsed
```

JobID	JobName	Partition	AllocTRES	CPUTime	TotalCPU	MaxRSS
28917	memBurn	normal	billing=8,cpu=8,mem=30G,node=1	00:08:16	00:57.790	
28917.batch	batch		cpu=8,mem=30G,node=1	00:08:16	00:57.788	15667884K
28919	memBurn	normal	billing=8,cpu=1,mem=20G,node=1	00:01:01	00:57.631	
28919.batch	batch		cpu=1,mem=20G,node=1	00:01:01	00:57.629	15667884K

Too much memory
and CPU requested.

Fixed it ! (only 20G of
memory and 1 CPU)

Validate your slurm script before running it

`sbatch --test-only jobScript.sh`

- Validates your script's allocation before running it.
- Gives estimate of when job will run.
- **Warning:** does not detect bad `--time` allocations.

`sbatchTest_withErrors.sh`

```
#!/bin/bash
#SBATCH --account=
#SBATCH --partition=normal
#SBATCH --job-name=testJob
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --mem=20T
#SBATCH --time=2-00:30:00
#SBATCH --export=NONE
```

```
[user@login1 ~]$ sbatch --test-only sbatchTest_withErrors.sh
sbatch: unrecognized option '--ntasks=1'
allocation failure: Invalid account or account/partition combination specified
allocation failure: Requested node configuration is not available.
```

... fix errors in script ...

```
[user@login1 ~]$ sbatch --test-only sbatchTest_withErrors.sh
sbatch: Job 28858 to start at 2019-06-07T17:21:55 using 2 processors on nodes
cpt002 in partition normal
```

```
[user@login1 ~]$ sbatch sbatchTest_withErrors.sh
Submitted batch job 28873
```

```
[user@login1 ~]$ squeue -u $USER
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
28873	normal	testJob	rengler	PD	0:00	1	(PartitionTimeLimit)

Slurm: other commands

	Slurm command
Inquire about job status	squeue / sacct
Force termination of running job	scancel
Display information about queues/partitions	sinfo
Display information about cluster nodes	sinfo --Node
Move job to a different queue/partition	scontrol

queue – monitor running jobs

- lists submitted and running jobs or job.steps, with their current status.
- By default *queue* will only show jobs that are PENDING [PD], RUNNING [R] or COMPLETING[CG].

General syntax:

***queue* <options>**

Example:

```
[user@login1 ~]$ queue -u $USER
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST (REASON)
1322	normal	testJob	rengler	R	1:20:03	1	cpt03

Job name

Abbreviated job status.
R=RUNNING

wall time since the
jobs started (H:M:S)

Node(s) running the job.

- Adding the **--long** / **-l** option expands the job's status and displays its runtime limit.

```
[user@login1 ~]$ queue --long
```

JOBID	PARTITION	NAME	USER	STATE	TIME	TIME_LIMIT	NODES	NODELIST (REASON)
1322	normal	testJob	rengler	RUNNING	0:03	2:30:00	1	cpt03

Expanded job status.

Time limit of job

Note: a custom time limit was here set with --time.

sacct – show completed jobs

- By default **sacct** will only show jobs that have finished (or are still running/pending) within the current day.

General syntax: **sacct <options>**

- To display older jobs, **--starttime=MM.DD[.YY]** can be used to specify the date after which jobs should be listed. Different formats are accepted for entering the date:

```
sacct --starttime=06.14      # June 6th of current year.  
sacct --starttime=0614      # same as above.  
sacct --starttime=06.14.19  # June 6th 2019.  
sacct --starttime=061419    # same as above.
```

Options: (non-exhaustive list)

long option	short	description
--format	-o	Customize the output's columns. E.g. this will add memory/CPU/time consumption: --format=jobid,jobname,partition,account%30,alloctres%40,cputime,totalcpu,maxrss,submit,state%15,exitcode,elapsed
--starttime	-S	Display jobs that finished after the selected date. Several date formats are recognized: MMDDYY, MMDD, MM.DD.YY, MM.DD, MM/DD/YY, MM/DD, YYYY-MM-DD.
--state	-s	Report only jobs matching the selected state(s). Important: when using this option, the --starttime option must also be specified (otherwise only running/pending jobs are shown)
--long	-l	Displays more info (warning: really a lot of info!).

sacct examples

```
[user@login1 ~]$ sacct
```

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
1062	echo	normal	vital-it	1	COMPLETED	0:0
1340	test2	long	vital-it	3	COMPLETED	0:0
1340.batch	batch		vital-it	3	COMPLETED	0:0
1339	test1	normal	vital-it	3	COMPLETED	0:0
1339.batch	batch		vital-it	1	COMPLETED	0:0
1339.0	printHost+		vital-it	1	COMPLETED	0:0
1339.1	printHost+		vital-it	1	COMPLETED	0:0
1339.2	printHost+		vital-it	1	COMPLETED	0:0
1360	test4	normal	vital-it	2	CANCELLED	0:0
1360.batch	batch		vital-it	1	CANCELLED	0:15
1360.0	samtools+		vital-it	1	CANCELLED	0:15
1360.1	samtools+		vital-it	1	CANCELLED	0:15
1361	test5	normal	vital-it	2	FAILED	1:0
1361.batch	batch		vital-it	1	FAILED	1:0
1363	memBurn	normal	vital-it	4	OUT_OF_MEMORY	0:125
1356.Batch	memBurn		vital-it	4	OUT_OF_MEMORY	0:125

"srun" job.

"sbatch" job.

"sbatch + srun"
with 3 tasks
(3 job steps).

Job cancelled
with "scancel".

Job failed.

Job terminated by slurm
because it exceeded
memory allocation.



Slurm exit codes explained: The "ExitCode" column has 2 numbers separated by a colon, e.g. "0:0", "1:0" or "0:125".

- first value is your script's exit code. E.g, 0 = no error, 1 = error.
- second value is slurm's internal exit code. E.g. 0 = no error, 15 = cancelled by user, 125 = exceeded memory.

To add useful information to the output of *sacct*, as job wall time, CPU and memory usage or submit time, the following command can be used:

sacct --format=jobid,jobname,partition,account%30,alloctres%40,cputime,totalcpu,maxrss,submit,state%15,exitcode,elapsed

```
[user@login1 ~]$ sacct --format=jobid,jobname,partition,account%30,alloctres%40,cputime,
totalcpu,maxrss,submit,state%15,exitcode,elapsed
```

JobID	JobName	Partition	Account	AllocTRES	CPUTime	TotalCPU
28903	cpuBurn	normal	cours_intro_hpc	billing=8,cpu=8,mem=4G,node=1	00:04:08	03:59.013
28903.batch	batch		cours_intro_hpc	cpu=8,mem=4G,node=1	00:04:08	03:59.011
28917	cpuBurn	normal	cours_intro_hpc	billing=8,cpu=8,mem=30G,node=1	00:08:16	00:57.790
28917.batch	batch		cours_intro_hpc	cpu=8,mem=30G,node=1	00:08:16	00:57.788
28919	memBurn	normal	cours_intro_hpc	billing=8,cpu=8,mem=20G,node=1	00:16:08	00:57.631
28919.batch	batch		cours_intro_hpc	cpu=8,mem=20G,node=1	00:16:08	00:57.629
28920	memBurn	normal	cours_intro_hpc	billing=8,cpu=8,mem=10G,node=1	00:07:52	00:56.106
28920.batch	batch		cours_intro_hpc	cpu=8,mem=10G,node=1	00:07:52	00:56.104

MaxRSS	Submit	State	ExitCode	Elapsed
	2019-06-13T09:33:37	COMPLETED	0:0	00:00:31
11596K	2019-06-13T09:33:37	COMPLETED	0:0	00:00:31
	2019-06-13T11:28:04	COMPLETED	0:0	00:01:02
4269804K	2019-06-13T11:28:04	COMPLETED	0:0	00:01:02
	2019-06-13T11:35:00	COMPLETED	0:0	00:02:01
15667884K	2019-06-13T11:35:00	COMPLETED	0:0	00:02:01
	2019-06-13T11:38:07	OUT_OF_MEMORY	0:125	00:00:59
4271244K	2019-06-13T11:38:08	OUT_OF_MEMORY	0:125	00:00:59

Column legend:

AllocTRES = requested resources.

Elapsed = wall time.

CPUTime = wall time * cpu requested.

TotalCPU = actual CPU usage time.

MaxRSS = maximum memory usage.

List of the most common slurm job states (non-exhaustive list)

Status	Short	Description
PENDING	PD	Job is awaiting resource allocation.
RUNNING	R	Job currently has an allocation.
COMPLETED	CD	Job has terminated all processes on all nodes with an exit code of zero.
FAILED	F	Job terminated with non-zero exit code or other failure condition.
TIMEOUT	TO	Job terminated upon reaching its time limit.
CANCELLED	CA	Job was explicitly canceled by the user or system administrator. The job may or may not have been initiated.
SUSPENDED	S	Job has an allocation, but execution has been suspended and CPUs have been released for other jobs.
COMPLETING	CG	Job is in the process of completing. This state should generally last only a few seconds, and if longer might be indicative of a problem.
OUT_OF_MEMORY	OOM	Job was terminated because it exceeded its memory allocation, i.e. it used more memory (RAM) than what was requested with the <code>--mem</code> option.

scontrol – switch job to a different partition

- *scontrol* can be used to modify the partition of a pending job – **but not of a running job !**
- Most *scontrol* commands are meant for sysadmins rather than regular users.



General syntax: `scontrol update jobid=<jobID> partition=<partition name>`

Example: Switching a PENDING job to the “long” partition.

```
[user@login1 ~]$ sbatch --partition=normal --time 3-00:00:00 jobScript.sh
Submitted batch job 28878

[user@login1 ~]$ squeue -l
JOBID PARTITION      NAME      USER      STATE  TIME  TIME_LIMIT NODES  NODELIST
28878   normal    testJob   rengler   PENDING 0:00   3-00:00:00  1  (PartitionTimeLimit)

[user@login1 ~]$ scontrol update jobid=28878 partition=long
[user@login1 ~]$ squeue -l
JOBID PARTITION      NAME      USER      STATE  TIME  TIME_LIMIT NODES  NODELIST
28878   long      testJob   rengler   RUNNING 0:23   3-00:00:00   1    cpt002
```

***scontrol** – doesn't work with running jobs...*

Warning: Running jobs cannot be switched anymore !

```
[user@login1 ~]$ sbatch --partition=normal --time 24:00:00 jobScript.sh
Submitted batch job 28878

[user@login1 ~]$ squeue -l
JOBID    PARTITION     NAME       USER      STATE   TIME   TIME_LIMIT  NODES    NODELIST
28878     normal    testJob    rengler    RUNNING 23:10   24:00:00     1       cpt005

[user@login1 ~]$ scontrol update jobid=28878 partition=long
Job is no longer pending execution for job 28877
```

partition switch failed!



Warning: Extending time limit requires admin privileges.

```
[user@login1 ~]$ sbatch --partition=normal --time 3:00:00 jobScript.sh
Submitted batch job 28878

[user@login1 ~]$ squeue -l
JOBID    PARTITION     NAME       USER      STATE   TIME   TIME_LIMIT  NODES    NODELIST
28878     normal    testJob    rengler    RUNNING 02:30   03:00:00     1       cpt005

[user@login1 ~]$ scontrol update jobid=28878 TimeLimit=24:00:00
Access/permission denied for job 28878
```

runtime limit extension failed!



scancel – terminate jobs and job steps

- *scancel* allows you to terminate jobs or job steps.
- Jobs and job steps can be terminated individually or by groups (e.g. all jobs of a given user in a given state).

General syntax:

`scancel <jobID>`
`scancel <jobID.step>`

Options: (non-exhaustive list)

long option	short	description
--user	-u	Kill all jobs for a given user.
--name	-n	Kill job(s) that match a given job name.
--state	-t	Kill all jobs in a given state for the current user. States can be given either by their full name, e.g. PENDING, RUNNING, SUSPENDED, or their short form, e.g. PD, R, S.

Examples:

```
[user@login1 ~]$ scancel 45002
```

```
[user@login1 ~]$ scancel 45001.1
```

```
[user@login1 ~]$ scancel -u bob --state PENDING
```

← Terminate job 45002

← Terminate step 1 of job 45005

← Terminate all jobs from user "bob" that are in state "pending"

sinfo – display info about nodes and partitions

General syntax:

sinfo
sinfo --Node

- *sinfo* displays the list of all partitions and nodes available on the cluster.
- By default, *sinfo* shows displays info in “partition oriented” format, where nodes are grouped by state and partitions.

```
[user@login1 ~]$ $ sinfo
```

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
normal*	up	1-00:00:00	2	fail	cpt[04,07]
normal*	up	1-00:00:00	5	idle	cpt[01-03,05-06]
long*	up	10-00:00:00	7	aloc	cpt[07-10,15,21-22]

Partition name

Availability up/down

Max runtime of partition

Node count

Node state:
idle = available to start job.
aloc = in use.
fail = node failed.
maint = under maintenance

Node list

- Adding the **--long / -l** option displas additional info:

```
[user@login1 ~]$ $ sinfo --long
```

PARTITION	AVAIL	TIMELIMIT	JOB_SIZE	ROOT	OVERSUBS	GROUPS	NODES	STATE	NODELIST
normal*	up	1-00:00:00	1-infinite	no	NO	all	1	fail	cpt04
normal*	up	1-00:00:00	1-infinite	no	NO	all	4	idle	cpt[01-03,05]

min-max number of nodes that can be requested per job

User groups that can submit jobs to partition. “all” = everyone.

sinfo – display info about nodes and partitions

- Use **sinfo --Node** / **sinfo -N** to display info on individual nodes (i.e., “Node oriented” format).

```
[user@login1 ~]$ $ sinfo -N
[user@login1 ~]$ $ sinfo --Node
```

NODELIST	NODES	PARTITION	STATE	CPUS	S:C:T	MEMORY	TMP_DISK	WEIGHT	AVAIL_FE	REASON
cpt01	1	normal*	idle	32	4:8:1	31950	48000	1	(null)	none
cpt02	1	normal*	idle	32	4:8:1	31950	48000	1	(null)	none
cpt05	1	normal*	idle	8	2:4:1	7800	115000	1	(null)	none

CPU count
on node

Socket : Core : Thread
Number of CPUs = S*C*T

RAM on
node [MB]

size of
/tmp [MB]

reason a node
is unavailable

- The **--nodes** / **-n** option can additionally be used to restrict display to certain nodes.

Warning: do not confuse the **--Node** and **--nodes** options.

```
[user@login1 ~]$ $ sinfo -N -n cpt[01-02]
[user@login1 ~]$ $ sinfo --Node --nodes cpt01,cpt02
```

NODELIST	NODES	PARTITION	STATE	CPUS	S:C:T	MEMORY	TMP_DISK	WEIGHT	AVAIL_FE	REASON
cpt01	1	normal*	idle	32	4:8:1	31950	48000	1	(null)	none
cpt02	1	normal*	idle	32	4:8:1	31950	48000	1	(null)	none

exercise 6

Slurm: job arrays

Job arrays – run the same job multiple times

What job arrays are...

➔ Job arrays are a way to replicate n times the exact same job.

➔ So essentially they replace a bash for loop like:

```
for x in $(seq 1 5); do
  sbatch jobScript.sh inputSample_${x}.fastq
done
```

with:

```
sbatch jobArrayScript.sh
```

➔ why bother ?

- Better reproducibility (all the info is in your script).
- Easier job management for you (e.g. you get single notification when the job is done).
- Better job scheduling by slurm = your jobs might be dispatched faster.

and what they are not...

➔ Array job != MPI (message passing interface) jobs.

➔ Individual jobs of an array are likely to run on different compute nodes, but they are fully independent (each with its own global allocation) and do not communicate.

➔ Resources requested for an array job (memory, CPU, ...) are the resources needed for one replicate of the array, not the entire array. E.g. if you run an array with 15 replicates, and each needs 20 GB of memory, you request is “--mem=20G” (and not --mem=300G).

how to submit job arrays ?

long option	short	description
<code>--array=<index list></code>	<code>-a</code>	<p>Submit a job array with indexes “index list”.</p> <p>The index list can be passed as ranges (a-b = from a to b), comma separated values (a,b,c = a, b and c) or a combination of both (a-b,c = from a to b, and c). Here are some examples:</p> <p><code>--array=1-25</code> : 25 replicates, with indexes from 1 to 25.</p> <p><code>--array=0-24</code> : 25 replicates, with indexes from 0 to 24.</p> <p><code>--array=9,12,23,45</code> : 4 replicates, with indexes 9, 12, 23 and 45.</p> <p><code>--array=1-5,7,9-10</code> : 8 replicates, with indexes 1,2,3,4,5,7,9,10.</p> <p>Note: index values must be positive integers.</p>



Custom step value:

Use the `:x` modifier to set the increase step within a range to “x”. For instance:

`--array=1-9:2` ==> run an array of 5 replicates with indexes 1, 3, 5, 7 and 9.

`--array=0-23:5` ==> run an array of 5 replicates with indexes 0, 5, 10, 15 and 20.



Limit number of running jobs:

Use the `%x` modifier to limit the number of simultaneously running jobs to “x” .

`--array=1-250%20` ==> This array has 250 jobs, but at most 20 will be running at anytime.

Job arrays – environment variables and filename patterns

Shell environment variables

- With job arrays, it is up to you to use the variable `$SLURM_ARRAY_TASK_ID` at some point in your bash script to modify the inputs/outputs of your job in each replicate of the array.
- `$SLURM_ARRAY_TASK_ID` takes the values of the indexes passed to the `--array` option. E.g. in a job with “`--array=1-5,9`”, the values of `$SLURM_ARRAY_TASK_ID` will be 1,2,3,4,5 and 9.
- The value of the array index (`$SLURM_ARRAY_TASK_ID`) is the only thing that varies between replicates within an array.

variable		expands to
most useful {	<code>\$SLURM_ARRAY_TASK_ID</code>	job array index ID number for the current array replicate.
	<code>\$SLURM_ARRAY_JOB_ID</code>	job array master ID value. In array jobs, this must be used instead of <code>\$SLURM_JOB_ID</code>.
	<code>\$SLURM_ARRAY_TASK_COUNT</code>	total number of tasks in a job array.
	<code>\$SLURM_ARRAY_TASK_MAX</code> <code>\$SLURM_ARRAY_TASK_MIN</code>	job array's maximum/minimum index number.
	<code>\$SLURM_ARRAY_TASK_STEP</code>	job array's index increase step size.

Filename patterns variables (to use in file names with the `--output/--error` options).

variable	expands to
<code>%A</code>	job array's master job ID number (unique job ID for all jobs in the array). %A must be used instead of %j in array jobs.
<code>%a</code>	job array index ID number for the current array replicate.

Job arrays – example

Replicate a same R script for 11 samples: “sample_1.txt”, “sample_2.txt”, ..., “sample_12.txt”.
Note that “sample_11.txt” is missing.

```
#!/bin/bash
#SBATCH --account=Piname_project
#SBATCH --array=1-10,12
#SBATCH --job-name=testArray
#SBATCH --partition=normal
#SBATCH --output=%x_%A-%a.out
#SBATCH --error=%x_%A-%a.err
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --mem=16G
#SBATCH --mail-user=user@unil.ch
#SBATCH --mail-type=ALL
#SBATCH --time=3:00:00
#SBATCH --export=NONE

# Commands...
cd scratch/wally/<Institution>/<faculty>/<department>/<PI>/<project_short_name>
module load Bioinformatics/Software/vital-it
module add R/3.5.1

mkdir arrayRun_${SLURM_ARRAY_TASK_ID}
Rscript myScript.R sample_${SLURM_ARRAY_TASK_ID}.txt
```

List of indexes over which to iterate.

Note the use of **%A** for jobID (instead of **%j** used in regular jobs). This value stays the same for all jobs within the array.

%a expands to the index value. This value changes with every job within the array.



Example output files:

testArray_28877_1.out

testArray_28877_1.err

testArray_28877_12.out

testArray_28877_12.err

\$SLURM_ARRAY_TASK_ID

expands to a different value for each job of the array.

Help and support

Software support - projects@vital-it.ch

- Questions / problems **related to software installation** (not on software usage itself).
- Software **installation / update** can sometimes take up to a few days.
- Use your **UNIL email address**.
- For questions / problem reports, try to be **as specific as possible** so we can reproduce your error.
 - software package and version?
 - commands leading to error.
 - input data and source directory. Check permissions to make sure the data is accessible to us.
 - slurm outputs/errors you have received.

General support – helpdesk@unil.ch

- For all other questions (e.g. **hardware** or **user account**).