Mila & DRAC cheat sheet

(formely Compute Canada)

by IDT team

Remember that every map is a simplification of reality.
This is a cheat sheet for Mila students using SLURM,
not a full tutorial, and also not a Linux/Git/PyTorch guide.
See **docs.mila.quebec/Cheatsheet.html** for pdf,
along with errata. Anticipate one update per year.
The complete up-to-date documentation at **docs.mila.quebec**.

PREPARED ON
2023-09-07
BEST BEFORE
2024-04-01

getting help

- search in `docs.mila.quebec`
- search for specific strings (or error messages) on the Mila slack
- visit `#mila-cluster` and `#compute-canada` (for DRAC)
- visit specific tool channels such as `#pytorch` and `#jax`
- go to the IDT office hours (Tuesday 3PM-5PM)
  by just walking into the IDT lab (room A.17) and saying hi
- open an IT support ticket by emailing **it-support@mila.quebec**
- contact DRAC support at `support@tech.alliancecan.ca`

milatools

Quick way to setup SSH to Mila cluster

```
pip install -U milatools; mila init
```

Open VSCode connected to an interactive session on **GPU** compute node

```
mila code /path/work [salloc arguments]
```

Inside VSCode you can also open a remote SSH to `mila-cpu` to automatically create an interactive session to a **CPU** node (configured by `mila init`). You can also `ssh mila-cpu` from a terminal to do the same.

Never run a program that takes more than a few seconds on a login node. Do not edit files remotely with VSCode directly on login nodes.

modules

`module avail`	Displays all the available modules
`module load <module>`	Loads <module>
`module spider <module>`	Shows details about <module>
`module load python/3.10`	Load python 3.10 to use it
`module load httpproxy`	Allows Wandb and Comet on DRAC

(NB: Many of those modules are outdated. Python 3.9 is semi-broken.)

SLURM commands

`salloc --gres=gpu:1 -c 2 --mem=12000`
Get an interactive job with one GPU, 2 CPUs and 12000 MB RAM

`sbatch`
Start a batch job (same options as salloc)

`sattach --pty <jobid>.0`
Re-attach a dropped interactive job

`sinfo`
Status of all nodes

`sinfo -O gres:27,nodelist,features -tidle,mix,alloc`
List GPU type and FEATURES that you can request

`savail`
List available gpu **(Mila only)**

`partition-stats [-v]`
Similar functionality to `savail` **(DRAC only)**

`scancel <jobid>`
Cancel a job

`squeue -u $USER`
Summary status of all YOUR active jobs

`squeue -j <jobid>`
Summary status of a specific job

`squeue -O jobid,name,username,partition,state,timeused,nodelist,gres,tres`
Status of all jobs including requested resources
(see the SLURM squeue doc for all output options)

`scontrol show job <jobid>`
Detailed status of a running job

`sacct -j <job_id> -o NodeList`
Get the node where a finished job ran

`sacct -u $USER -S <start_time> -E <stop_time>`
Find info about old jobs

`sacct -oJobID,JobName,User,Partition,Node,State`
List of current and recent jobs

sbatch / salloc commands

`-n, --ntasks=<number>`	Number of task in your script, usually =1
`-c, --cpus-per-task=<ncpus>`	Number of cores for each task
`-t, --time=<time>`	Time requested for your job
`--mem=<size[units]>`	Memory requested for all your tasks
`--gres=<list>`	Select generic resources such as GPUs: `--gres=gpu:GPU_MODEL`
`-p, --partition=<name>`	Partition for resource sharing (Mila cluster only)
`--account=<name>`	DRAC allocation for resources (DRAC only)
`-x, --exclude=<nodes>`	Exclude certain nodes from job submission

sbatch script example

```
#!/bin/bash
#SBATCH --ntasks=1                   # Default 1 task, optional
#SBATCH --partition=unkillable       # Ask for unkillable job
#SBATCH --cpus-per-task=2            # Ask for 2 CPUs
#SBATCH --gres=gpu:1                 # Ask for 1 GPU
#SBATCH --mem=10G                    # Ask for 10 GB of RAM
#SBATCH --time=3:00:00               # The job will run for 3 hours
#SBATCH -o /network/scratch/<u>/<username>/slurm-%j.out

# Load the required modules
module --quiet load anaconda/3
# Load your environment
conda activate "<env_name>"
# Copy your dataset on the compute node
cp /network/datasets/<dataset> $SLURM_TMPDIR
# Launch your job, tell it to save the model in $SLURM_TMPDIR
# and look for the dataset into $SLURM_TMPDIR
python main.py --path $SLURM_TMPDIR --data_path $SLURM_TMPDIR
# Copy whatever you want to save on $SCRATCH
cp $SLURM_TMPDIR/<to_save> /network/scratch/<u>/<username>/
```

multi-GPU, multi-node

See `docs.mila.quebec/examples/distributed/index.html` for minimalist standalone code.

1 node with 1 GPU

```
#SBATCH --gpus-per-task=rtx8000:1
#SBATCH --cpus-per-task=4
#SBATCH --ntasks-per-node=1
#SBATCH --mem=16G
#SBATCH --time=00:15:00
```

1 node with 4 GPUs

```
#SBATCH --gpus-per-task=rtx8000:1
#SBATCH --cpus-per-task=4
#SBATCH --ntasks-per-node=4
#SBATCH --mem=16G
#SBATCH --time=00:15:00
```

2 nodes with 4 GPUs each

```
#SBATCH --gpus-per-task=rtx8000:1
#SBATCH --cpus-per-task=4
#SBATCH --ntasks-per-node=4
#SBATCH --nodes=2
#SBATCH --mem=16G
#SBATCH --time=00:15:00
```

If you have N parallel jobs that each require 1 GPU, don't try to schedule them in a multi-GPU way. Submit many separate jobs, maybe use job arrays, or consider packing many experiments in a single job with 1GPU.

checkpointing, profiling, scaling

Powerful GPUs cost approximately $1/h when amortized over their expected life.
If you use only one GPU for active development, it's acceptable to be inefficient.
Consider using a less powerful GPU or a "MIG" instance if possible.

Things change when you run large-scale experiments. You need to
- profile your code to make sure you properly use the GPUs allocated (i.e. "GPU Utilization"),
- use checkpoints properly to resume your experiments when they crash or get preempted.

Easy ways to measure "GPU Utilization" include Wandb, `nvidia-smi` and the DRAC "portail".
See also **docs.mila.quebec/examples/good_practices/checkpointing/index.html** for an example of proper checkpointing.

Research involves exploring and testing ideas that don't necessarily work out in the end.
This is a good use of the cluster when done properly. Mila is a research institute.

Don't be the researcher who runs 200 jobs each running for 24h and using only 2% of a GPU.
They've just wasted $5000. Lack of proper checkpointing can lead to same levels of waste.
Avoid grid search for hyperparameter optimization. Better tools reduce unnecessary computation.

Mila

docs.mila.quebec	official docs
dashboard.server.mila.quebec	node and GPU monitoring
datasets.server.mila.quebec	datasets already shared on cluster
clockwork.mila.quebec	dashboard for jobs (currently in beta)

The Mila cluster is available for **all students supervised by a Mila core prof** and for Mila employees.
Not MsPro students, nor students of non-core Mila profs. Exceptions exist.

		GPU	GPU		CPU				
Node Name	**Qty**	**N**	**Model**	**Mem (GB)**	**Cores**	**Mem (GB)**	**Tmp (TB)**	**SLURM features**	**optimal ratios GPU:CPU:RAM**
GPU compute nodes									
cn-a[001-011]	11	8x	RTX8000	48	40	384	3.6	turing,48gb	1 : 5 : 48GB
cn-b[001-005]	5	8x	V100	32	40	384	3.6	volta,nvlink,32gb	1 : 5 : 48GB
cn-c[001-040]	40	8x	RTX8000	48	64	384	3	turing,48gb	1 : 8 : 48GB
cn-g[001-029]	29	4x	A100	80	64	1024	7	ampere,nvlink,80gb	1 : 16 : 256GB
cn-i001	1	4x	A100	80	64	1024	3.6	ampere,80gb	1 : 16 : 256GB
cn-j001	1	8x	A6000	48	64	1024	3.6	ampere,48gb	1 : 8 : 128GB
DGX Systems									
cn-d[001-002]	2	8x	A100	40	128	1024	14	ampere,nvlink,dgx,40gb	1 : 16 : 32GB
cn-d[003-004]	2	8x	A100	80	128	2048	28	ampere,nvlink,dgx,80gb	1 : 16 : 64GB
cn-e[002-003]	2	8x	V100	32	40	512	7	volta,nvlink,dgx,32gb	1 : 5 : 16GB
CPU compute nodes									
cn-f[001-004]	4	-	-	-	32	256	10	rome	0 : 1 : 8GB
cn-h[001-004]	4	-	-	-	64	768	7	milan	0 : 1 : 12GB
Legacy GPU compute nodes									
kepler5	1	2x	V100	16	16	256	3.6	volta,16gb	1 : 8 : 128GB

partition name	max resource usage	max time	note
`unkillable`	6 CPUs, mem=32G, 1 GPU	2 days	
`unkillable-cpu`	2 CPUs, mem=16G	2 days	CPU-only jobs
`short-unkillable`	24 CPUs, mem=128G, 4 GPUs	3 hours (!)	
`main`	8 CPUs, mem=48G, 2 GPUs	5 days	
`main-cpu`	8 CPUs, mem=64G	5 days	CPU-only jobs
`long`	no limit of resources	7 days	
`long-cpu`	no limit of resources	7 days	CPU-only jobs

Jobs on the Mila cluster are all *preemptible*, except for those in the `unkillable` partitions. This means that they can be terminated and requeued automatically to allow higher-priority jobs to run. There is a very limited number of jobs that can run in `unkillable` partitions.

Partitions are specified with the `--partition` flag (may be obsolete in 2024).

path	storage/inodes	speed	backup?	mounted
`$HOME`	100GB / 1M	low	yes	all nodes
`$SCRATCH`	-	medium	no	all nodes
`$SLURM_TMPDIR`	-	high	no	`cn-*`
`/network/projects`	varies	medium	no	all nodes
`/network/datasets`	read-only	high	no	all nodes
`/network/weights`	read-only	high	no	all nodes
`$ARCHIVE`	500GB	low	no	`login-*`

Use `disk-quota` to see your current usage of storage (`$HOME` only).

MIG (a fractional part of a powerful GPU) `--gres=gpu:a100l.2`

`a100l.2g.20gb` `a100l.2`	A100 (20GB, 2/7 of compute) `sbatch --gres=gpu:a100l.2 -c=4 --mem=32G ...`	48 available
`a100l.3g.40gb` `a100l.3`	A100 (40GB, 3/7 of compute) `sbatch --gres=gpu:a100l.3 -c=8 --mem=64G ...`	48 available
`a100l.4g.40gb` `a100l.4`	A100 (40GB, 4/7 of compute) `sbatch --gres=gpu:a100l.4 -c=8 --mem=64G ...`	24 available

Use `savail` to list the GPUs available. Alternatively, go to **dashboard.server.mila.quebec**.

MIG instances should never be used for multi-GPU training. It is slow and absurd compared to using a single full GPU. For up-to-date tips to avoid hitting a MIG node in your multi-GPU training, refer to either **docs.mila.quebec** or the **#mila-cluster** Slack channel. SLURM features (e.g. `--constraint=<something>`) will soon support this more elegantly.

DRAC

docs.alliancecan.ca/wiki
portail.narval.calculquebec.ca/secure/jobstats/<username>/<jobid>
portail.beluga.calculquebec.ca/secure/jobstats/<username>/<jobid>

DRAC (formely know as Compute Canada) offers access to compute clusters to all researchers in Canada. Students first have access to their supervisor's "default" allocation. Additionally, **all students supervised by a Mila core prof** and all Mila employees can be added to a "mega allocation" under Yoshua Bengio's name, allowing access to even more resources. To create your initial account with DRAC, see **https://docs.mila.quebec/Extra_compute.html#account-creation**.

	(shared mega-allocation)		(your supervisor's default allocation)					
	rrg-bengio-ad_**gpu**	rrg-bengio-ad_**cpu**	def-yourprof-**gpu**	def-yourprof-**cpu**	GPU types	unrestricted internet?	Wandb?	Comet?
narval	**154**	**917**	3	**195**	A100	no	httpproxy (limited)	httpproxy
beluga	**127**	**197**	4	63	V100	no	httpproxy (limited)	httpproxy
cedar	**127**	**197**	3	71	P100, V100	yes	yes	yes
graham	**0**	**0**	3	40	P100,V100,T4	no	no	no

The rrg- values above are guaranteed resources for the year, while the def- values are an estimate of the best effort to share excess resources available across Canada.

DRAC compute nodes have similar roles as the Mila cluster for `$HOME`, `$SCRATCH` and `$SLURM_TMPDIR`.
See also `$HOME/projects/<account>/<your_username>`. Use `diskusage_report` to see your usage.

1. Contrary to the Mila cluster, DRAC allocations have a single *fair share* value for all users under the account. Jobs that are the easiest to run will run first, no matter to whom they belong.
This might not feel fair. IDT does not control this. DRAC does.

Don't be a bad actor by submitting 1k small jobs because you will end up monopolizing the clusters to the detriment of everyone.
This is the price for DRAC clusters having preemption disabled.

If your jobs are queued and never run, try to make them more appealing to the scheduler by asking for optimal resources.

2. You have access to "default accounts" that starts with a "def-".
These are shared between members of your research group instead of the whole Mila. This resources are underused. Free CPUs/GPUs!

3. The shared storage on DRAC is particular because we run out of inodes (i.e. number of files) faster than the actual storage space.

4. IDT does not admin rights on DRAC clusters.

jobs that the scheduler likes

time bins	<=3h	<=12h	<=24h	<=72h	<=168h

GPU:CPU:RAM ratios

Beluga	1 : 10 : 46G x4	
Narval	1 : 12 : 123G x4	
Cedar	1 : 8 : 46G x4	(V100 32GB)
	1 : 6 : 31G x4	(P100 12GB)
	1 : 6 : 62G x4	(P100 16GB)

DRAC clusters use a different method to queue jobs. You cannot specify a `--partition`.
Jobs fall in "bins" based on time and resources requested. Ask for things that are easy to schedule, the scheduler will be much nicer to you. If you break your 12h job into 4 chunks of 3h (with checkpointing), you will get resources more easily. If you ask for 13h, you will be put into the `<=24h` bin, which is not advantageous to you.

If you ask for certain ratios of GPU:CPU:RAM when submitting jobs, the scheduler will also favor you.