IDRIS Cheat Sheet

Guide to Linux, module, SLURM & IDRIS commands

Accessing JEAN ZAY - HPE SGI 8600

Logging in from a terminal
ssh *login*@jean-zay.idris.fr

PuTTY	SSH client for Windows
Xming/XQuartzX11	X11 display server for Windows/Mac

Transferring files between JEAN ZAY and your system
sftp *login*@jean-zay.idris.fr destination
sftp> put *localSource JZDestination*
sftp> get JZSource localDestination

scp localSource *login*@jean-zay.idris.fr:*JZDestination*
scp *login*@jean-zay.idris.fr:JZSource localDestination

for large file (larger than 2GB) transfer
bbftp -e 'put localSource *JZDestination*' -u login -s jean-zay.idris.fr
bbftp -e 'get JZSource localDestination' -u login -s jean-zay.idris.fr

FileZilla	SFTP client for Linux, Windows, Mac
WinSCP	SFTP/SCP client for Windows
BBFTP In2p3	BBFTP client for Linux, Mac

Basic Linux file management

man *command*	Display the manual page for command
pwd	Print out the present working directory path
cd *dir*	Change directory to *dir*
~ (tilde)	Home directory
. (period)	Current directory
.. (double dot)	Parent directory
ls	List the files in the current directory
ls -lh	Show long listing of files
ls -a	Show hidden files as well
rm *file*	Delete *file*
mkdir *dir*	Create empty directory called *dir*
rm -r *dir*	Remove directory dir and **all of its content**
cp *file1 file2*	Copy *file1* into *file2*
cp *file dir*	Make a copy of file *file* inside directory *dir*
mv *file1 file2*	Rename file *file1* as *file2*
mv *file dir*	Move file *file* inside directory *dir*
wget *URL*	Download a file from *URL* in Internet - Only reachable from a frontal node
git clone *URL*	Clone a git repository - Only reachable from a frontal node
unzip *file*.zip zip -r *file*.zip *dir*	Extract / Compress a ZIP file
tar -xzf *ftgz* tar -czf *ftgz dir*	Extract / Compress a gz compressed tarball (*ftgz*)
chmod	Change read/write/execute permissions
setfacl *ACLOPT dir*	Define Access Control List for *dir*
which *cmd*	List the full file path of a command
whereis *cmd*	List binary/source/manual paths of *cmd*
du -hd1 *dir*	List size of directory *dir* and its subdirectories
du -inodes -d1 *dir*	List inodes amount of directory *dir* and its subdirectories
find	Find files in a directory
tree	Display recursive directory listing

System information

hostname	Show device/node name
date	Return date time of system
top	Display processor and memory usage
htop	Display an augmented *top* viewer
watch nvidia-smi	Provides monitoring information for GPUs

Viewing and editing text files

cat *file*	Print entire content of file
more *file*	Incrementally display content of file
less *file*	Similar to more, but with additional features
head *file*/tail *file*	Display *file* header / footer
emacs	Extensible and customizable text editor
vim	Minimalist yet powerful text editor
nano	Simple text editor

Aliases and system variables

alias	Automatically replace one command with another
env	List all system variables
export *var*=*val*	Create environment variable *$var* and set value to *val*
echo	Print text to the screen
echo $*var*	Print the value of variable *var*

Input and output redirection

$(*command*)	Run the command inside $() first, then insert the output to the rest of the command. Same as `` `command` `` with grave marks.
<	Standard input redirection
>	Standard output redirection
2>	Standard error redirection
2>&1	Standard error and output redirection
cmd1 | *cmd2*	Pipe command; redirects output of command *cmd1* to input of *cmd2*
tee *file*	Read standard input and writes it to standard output and a file *file*, ex: *echo "0" | tee file.txt*
eval *args*	Execute arguments as a shell command

Filters

wc	Word count
grep	Get a regular expression and prints it
sort	Sort input
uniq	Filter duplicate lines
cut	Cut specific fields or columns
sed	Stream editor for search and replace
awk	Extensive tool for complex filtering tasks

Modules management

`module avail` *firstLetterName*	Show all available module names fitting with *firstLetterName* string
`module show` *name*	Display module information
`module load` *name*	Load module in the environment
`module list`	List currently loaded modules
`module unload` *name*	Remove module from environment
`module purge`	Remove all modules

SLURM

`squeue -lu $USER`	Show job queue for user.
`squeue -u $USER --start`	Show pending jobs and the estimated start time for user *idrisid*
`scontrol show job` *jobid*	Display control states for a job
`scontrol show node` *nodeid*	Display control states for a node
`scancel` *jobid*	Delete the job with *jobid*
`sinfo`	Display the system settings
`sacct`	Display job accounting information
`sbatch` *filename*	Submit a batch script *filename*
`salloc`	Obtain an interactive job allocation
`srun` *execfile*	Obtain a job allocation and execute an application

SBATCH directives

`--job-name=`*name*	Job name
`--account=`*my_project@dev*	Account to charge. The *dev* field could be *cpu* or *gpu*
`--partition=`*name*	Submit to partition *name*
`--qos=`*name*	Subscribe to a Quality of service
`--time=`*min*	Time limit; either *min* or *dd-hh:mm:ss*
`--nodes=`*count*	Number of nodes
`--ntasks-per-node=`*count*	Processes per node
`--ntasks=`*count*	Total processes
`--cpus-per-task=`*count*	CPU cores per process
`--gres=gpu:`*count*	GPUs per nodes
`--hint=nomultithread`	Turn off the *hyperthreading*
`--exclusive`	Allocate nodes in exclusive mode
`--output=`*file*	Standard output; defaults to *slurm-jobid.out* if omitted
`--error=`*file*	Write standard error to *file*
`--array=`*arrayspec*	Submit a collection of similar jobs
`--dependency=`*state*`:job`	Job dependency for multi-step and cascade jobs. state ex: *afterok*
`--mail-user=`*email*	Email for job alert
`--mail-type=`*type*	Email alert types: BEGIN, END, FAIL, REQUEUE, ALL
`-C, --constraint=`*feature*	Require specific feature on a node

SLURM Environment Variables

`$SLURM_JOBID`	Job ID
`$SLURM_JOB_NODELIST`	Names of nodes allocated to job
`$SLURM_NNODES`	Number of nodes allocated to job
`$SLURM_JOB_CPUS_PER_NODE`	CPU cores per node allocated to job
`$SLURM_NTASKS`	Number of tasks allocated to job
`$SLURM_SUBMIT_DIR`	Job submission directory
`$SLURMD_NODENAME`	Name of the node running the job script
`$SLURM_PROCID`	The MPI rank of the current process
`$SLURM_ARRAY_JOB_ID`	Job array's master job ID number
`$SLURM_ARRAY_TASK_ID`	Task id within job array
`$SLURM_ARRAY_TASK_COUNT`	Total number of tasks in a job array
`$SLURM_ARRAY_TASK_MIN`	Job array's min. ID (index) number
`$SLURM_ARRAY_TASK_MAX`	Job array's max. ID (index) number

IDRIS-specific commands

`idrdoc`	Show Jean Zay configuration infos
`idrquota`	Show quota disk infos. *-m | -w | -s (home/work/store)*
`idracct`	Indicate the CPU and/or GPU hours allocations.
`idrproj`	Display the projects or change the default project.
`idrenv`	Generate the environment shell commands for the current login
`idrjup`	Launch Jupyter notebook on the current node
`idrlab`	Launch Jupyter lab on the current node
`idr_compuse`	Verify the consumption status of your project

Tricky commands

`time ./my_exe`	Print the running time of an executable file
`eval $(idrenv -d projet)`	Force to switch to another of your projects.
`ssh nodeid nvidia-smi -l 1`	Display on frontal node, the monitoring information for GPUs for a specific node.
`ssh nodeid` `[nodeid ~]$ htop`	Display processor and memory usage for a specific node.
`sacct -A acc@gpu -S MMDDYY -E MMDDYY --format Elapsed,Submit,NNodes,JobID,NCPUS,NTasks,AllocGRES,User`	Display job's history for *account* between start date (-S) and end date (-E)
`srun --pty --sbatch_options bash`	Obtain a terminal on a compute node (e.g. Jupyter Notebook or Jupyter Lab launching on a node)

Tools (available from *module load*)

`idrmem`	Display the max consumptions of virtual and physical memory used by an MPI code.
`vtune`	Intel advanced profiling tool
`map`	Arm advanced profiling tool
`ddt`	Arm advanced debugging tool
`scalasca`	MPI and OpenMP profiling tool

Sbatch script examples

Mixed MPI/OpenMP parallel job

```
#!/bin/bash
#SBATCH --job-name=Hybride                  # Job name
#SBATCH --ntasks=8            # Number of MPI process
#SBATCH --cpus-per-task=10# Number of OpenMP threads
#SBATCH --hint=nomultithread     # no hyperthreading
#SBATCH --time=00:10:00              #  Max exec time
#SBATCH --output=Hybride%j.out        # out file name
#SBATCH --error=Hybride%j.out       # error file name

# submission directory
cd ${SLURM_SUBMIT_DIR}

# module cleaning
module purge

# module loading
module load intel-all/19.0.4

# echo launched commands
set -x

# number of OpenMP threads
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
# binding OpenMP
export OMP_PLACES=cores

# code exec
srun ./exec_mpi_omp
```

Multi-GPU Multi-node job execution

```
#!/bin/bash
#SBATCH -job-name=multi_gpu_multicore        # Job name
#SBATCH --partition=gpu_p13            # GPU partition
#SBATCH --ntasks=8    # nbr of MPI tasks (= nbr of GPU)
#SBATCH --ntasks-per-node=4    # nbr of task per node
#SBATCH --gres=gpu:4              # nbr of GPU per node
#SBATCH -cpus-per-task=10        # nbr of CPU per task
#SBATCH --hint=nomultithread    # no hyperthreading
#SBATCH --time=00:10:00         #  Max exec time
#SBATCH --output=multi_gpu_mpi%j.out    # out file name
#SBATCH --error=multi_gpu_mpi%j.out   # error file name

# module cleaning
module purge

# module loading
module load ...

# echo launched commands
set -x

# code exec
srun ./executable_multi_gpu_mpi_cuda-aware
```

Multi-GPU Mono-node job execution for AI

```
#!/bin/bash

#SBATCH --job-name="GTSRB Full conv."       # Job name
#SBATCH --ntasks=4        # nbr of tasks (= nbr of GPU)
#SBATCH --gres=gpu:4          # nbr of GPU per node
#SBATCH --cpus-per-task=10  # nbr of cores per task
#SBATCH --hint=nomultithread         # physical core
#SBATCH --time=03:00:00              #  Max exec time
#SBATCH --output="_batch/G.out"      # out file name
#SBATCH --error="_batch/G.err"       # error file name
#SBATCH --mail-user=mail@domaine
#SBATCH --mail-type=ALL

MODULE_ENV="tensorflow-gpu/py3/2.2.0"
RUN_DIR="$WORK/fidle/GTSRB"
RUN_SCRIPT="./run/full_convolutions.py"


# ---- Welcome...
echo '----------------------------------------'
echo "Start : $0"
echo '----------------------------------------'
echo "Job id         : $SLURM_JOB_ID"
echo "Job name       : $SLURM_JOB_NAME"
echo "Job node list  : $SLURM_JOB_NODELIST"
echo '----------------------------------------'
echo "Script         : $RUN_SCRIPT"
echo "Run in         : $RUN_DIR"
echo "With env.      : $MODULE_ENV"
echo '---------    --------------------------'
# ---- Module

module purge
module load "$MODULE_ENV"

# ---- Run it...
#
cd "$RUN_DIR"
srun python "$RUN_SCRIPT"
```

Storage disks

`$HOME`	Storage of configuration files, 3GB / 150k inodes, backed up
`$WORK`	Storage of source codes and input/output data, 5TB/500k inodes, backed up
`$SCRATCH`	SSD storage, 2.2PB shared by all users, unused cleaning, not backed up
`$JOBSCRATCH`	SSD storage, temporary execution directory specific to batch jobs, not backed up
`$STORE`	Long-term archive storage, 50TB / 100k inodes, not backed up

Shared by all project members disks

`$ALL_CCFRWORK`	Common *WORK* storage
`$ALL_CCFRSCRATCH`	Common *SCRATCH* storage
`$ALL_CCFRSTORE`	Common *STORE* storage

Public disk

`$DSDIR`	Large public datasets storage for AI

Partition names

`cpu_p1`	1528 x CPU 40-cores nodes (default for CPU jobs)
`gpu_p13`	default for GPU jobs
`v100-32g`	261 nodes 4-GPU(32GB) 40-cores
`v100-16g`	351 nodes 4-GPU(16GB) 40-cores
`gpu_p2`	Reserved for AI, 31 x 8-GPU(32GB) 24-cores nodes
`gpu_p2l`	Subpartition 11 nodes with 768 GB RAM CPU
`gpu_p2s`	Subpartition 20 nodes with 384 GB RAM CPU
`prepost`	4 pre/post processing nodes - 20h max - free of hour charge
`visu`	5 visualization nodes - 1h max - free of hour charge

Quality of Service names

`qos_cpu-t3`	CPU QOS (default)	20h, 512 nodes per job, 1200 nodes per user
`qos_cpu-t4`	CPU Long time job QOS	100h, 4 nodes per job, 32 nodes per *user*, 128 per qos
`qos_cpu-dev`	Dev. CPU QOS	2h, 128 nodes per job, 128 nodes per user, 1200 per qos
`qos_gpu-t3`	GPU QOS (default)	20h, 512 GPUs per job, 1024 GPUs per user
`qos_gpu-t4`	GPU Long time job QOS	100h, 16 GPUs per job, 128 GPUs per *user*, 512 per qos
`qos_gpu-dev`	Dev. GPU QOS	2h, 32 GPUs per job, 32 GPUs per *user*, 512 GPUs per qos