# Exercises

# HPC slurm exercises

**IMPORTANT:** for the course (and only for the course), please add the `--reservation=HPC-course --account=rfabbret_cours_hpc` option to all your `srun` and `sbatch` commands. This will allow your jobs to get processed faster.

## Exercise 1: make yourself at home

**Objectives:** in this exercise you will learn how to:

- login to the UNIL cluster
- find the "projects" you are associated with.
- locate the "/scratch" directories associated with your projects.
- copy files from your local machine to the cluster or download them from the internet.

**Tasks:**

1. Open a command line terminal on your local machine: a UNIX shell on GNU-Linux/Mac, Windows Subsystem for Linux or PuTTY on Windows.
2. Login to the wally cluster front-end using the command below. Note that you have to replace `<user_name>` with your actual UNIL user name. You will also be asked for your UNIL password.
   ```
   ssh <user_name>@wally-front1.unil.ch
   ```
3. List the projects that you are associated with using the following command:
   ```
   sacctmgr show user $USER WithAssoc format=User,Account%30,DefaultAccount%30
   ```
4. Locate the "/scratch" directory associated with your project(s) by browsing `/scratch/wally` using the `ls` command. The "/scratch" directory of your project is a directory where you have read and write permission. In principle, the path to the "/scratch" directory associated with your project(s) looks something like this:
   ```
   /scratch/wally/FAC/<Faculty>/<Department>/<PI name>/<project name>
   ```
   Replace `<Faculty>`, `<Department>` and `<PI name>` with your own values:
   - *Faculty*: FBM, FGS, FGSE, FTSR or Lettres.

    - *Department*: the department to which you are affiliated.
    - *PI name*: the user name of your supervisor.

5. Once you found your "/scratch" directory, make sure that you have read/write access on it:
   `ls -ld <your scratch directory>`
   The permissions on the directory should look like: `drwxrws---`
6. Move into your scratch directory and create a new directory named after your user name.
   `cd <your scratch directory>`
   `mkdir $USER`
   `cd $USER`
7. On your local machine, using your web-browser, download the exercise data from:
   `ftp://ftp.vital-it.ch/edu/HPC/HPC_exercises.tar.gz`
8. Copy the "HPC_exercises.tar.gz" file from your local machine to the cluster. This can be done either with UNIX commands ("scp", "rsync") or using a ftp program such as FileZilla.
   Here is the general syntax for using "scp" (note: this command must be executed on your *local* machine):
   `scp <file to upload> <user_name>@wally-front1.unil.ch:</path/to/directory/where/to/copy/file>`

   Replace: `<file to upload>` with the tarball file.
   `<user_name>` with your UNIL user name.
   `</path/to/directory/where/to/copy/file>` with the path of your own directory you just created (see point 6 above) on "/scratch/wally".
9. Back on the cluster, decompress the files that will be needed for the exercises.
   `tar -zxvf HPC_exercises.tar.gz`

**Note:** If you are unable to copy the `HPC_exercises.tar.gz` file from your local machine to the cluster, you can also download it directly from the cluster's front-end machine with the following command:
`wget ftp://ftp.vital-it.ch/edu/HPC/HPC_exercises.tar.gz`

---

---

## Exercise 2: search for software applications installed on the cluster

**Objective:** learn how to search for software availability on the cluster using either your web browser or the command line interface of the GNU-Linux shell on the cluster.

### A) search for software using your web browser

**Tasks:** go to the Vital-IT "bioinformatics tools" webpage [https://www.vital-it.ch/services/software] and browse/search through the different software categories. Answer the questions below.

**Questions:**

- How many versions of "samtools" are currently available?
- What is the category under which "samtools" is found?
- What is the command to load "samtools" version 1.4?

**Additional question:**

- if you have some application in mind that you plan to use, search whether it is installed on the cluster. If yes, what is the "module add" command to load that software?

## B) search for software from the command line, using the "vit_soft" command

**Tasks:**

1. list the content of the `/software/module` directory and browse through the different categories and sub-categories.
   *Reminder:* the UNIX command to list the content of a directory is `ls` or `ls -l`.
2. Since software might not always be in the category you expect, a more efficient way to find software from the command line is to use the `vit_soft` command. Note that before you can use this command, you must first load the software stack:
   `module add Bioinformatics/Software/vital-it`

   **Reminder**: here are some useful options for the `vit_soft` command:

   - `vit_soft -h`: display help for the command.
   - `vit_soft -s <string to search in package name>`: search for a software package based on its name. The search is case insensitive but will only match elements from the package name.
   - `vit_soft -b <name of binary file>`: search for a software package based on the exact name of a binary files it contains. Only exact matches are found and the search is case sensitive.
   - `vit_soft -m <string to search in package name>`: displays the module add command for a package based on its name. Only exact matches are found and the search is case sensitive.

**Questions:** using the vit_soft command, answer the questions below.

- Is "matlab runtime" available on the cluster? If yes, what is the command to load it?
- What are the different versions of the "R" software available on the cluster?
- Which package(s) provide the binary "Rscript"?

**Notes:** Almost all software available on the UNIL cluster is located in `/software` and the module files are found under `/software/module/<category>`.
The general directory structure in the cluster software stack is the following:
`/software/<category>/<package>/<version>`

---

---

# Exercise 3: load software from the cluster "software stack"

**Objective:** learn how load software from the cluster's software stack.

**Tasks:**
In this exercise, we want to run a really short R script. Because this script is very simple and takes less than 1 second to run, we are allowed to run it directly on the front-end machine. But keep in mind that anything longer than a couple of seconds is not allowed to run on the front-end!

Here are the steps to run the R script:

1. Change directory into `../exercise_3`, where you should find a script named "display_R_version.R".
2. Load the software stack with the command: `module add Bioinformatics/Software/vital-it`
3. Load R version 3.5.1 using the appropriate `module add` command.
4. Run the command:
   `Rscript display_R_version.R`
   **hint:** if the script displays a warning, make sure you loaded R 3.5.1.
   **Reminder:** here is a list of the most useful "module" commands that you might need for this exercise:

   - `module avail` : list all available modules.
   - `module list` : list all currently loaded modules.
   - `module add <path/to/module` : load the specified module.
   - `module rm </path/to/module>` : remove the specified module.
   - `module purge` : remove all loaded modules.

   **Question:** what is the nickname of R 3.5.1?

5. Now run the R script again, but this time with with R version 3.4.2.

**Questions:**

- What is the nickname of R 3.4.2?
- how did you switch to another version of R?
- how does the `module add` command affect the shell variable `$PATH` ?
- how many modules do you currently have loaded? What is the module command to check this?.
- what command(s) would you execute to revert from R 3.4.2 to R 3.5.1?

---

---

## Exercise 4: submit a job with "srun"

**Objective:** submit a simple job to get familiar with slurm's `srun` command. Understand the difference between the front-end machine - from where jobs are submitted -, and cluster nodes - where jobs are executed.

**Tasks:**

1. Change directory into `../exercise_4`, where you should find a shell script named "simple.sh".

2. Look at the content of the "simple.sh" script (e.g. using the `cat` or `less` command). As its name suggests, the "simple.sh" script is extremely simple and does only one thing: it displays the name of the host executing the script on the standard output.

3. Try to run the script directly on the front-end machine with the command: `./simple.sh`
   Note that in general you are not allowed to directly run commands on the front-end machine like what you just did. But since the "simple.sh" script is really small and runs in less than one second we get a pass for this time.

   **Questions:**

   - Why did the command ./simple.sh fail?

   - How can the problem be fixed? (**hint:** use the interpreter used in simple.sh to call the script).

4. Submit the "simple.sh" script to the cluster using slurm's `srun` command. Remember that one difference between `srun` and `sbatch` is that `srun` only accepts executable scripts.
   **hint:** to make a script executable, use the `chmod a+x <script name>` command.

**Questions:**

- why is the result different than when we executed the script on the front-end machine? Is this expected?
- what is the name of the machine that executed your simple.sh script?
- compare your result to those obtained by your neighbors. Are they the same? Should they be the same?

---

---

## Exercise 5: submit a job with "sbatch"

**Objective:** lean how to write a slurm job submission script, and how to submit jobs to the cluster using the `sbatch` command.

**Tasks:**

1. Change directory into `../exercise_5`, where you should find a small text file named "protein_sequence.seq". This file contains the sequence of a protein, i.e. the coded list of amino-acids that compose the protein.
2. You are now asked to write a script that will compare the protein sequence found in "protein_sequence.seq" against a database in order to determine from which organism the sequence originates. This can be done with a single command:
   `blastp -db "swiss" -query protein_sequence.seq`

   Note that you do not need to precisely understand the above command. You can simply copy-paste it to your script. But remember that before you can use the "blastp" command, you will need to load the corresponding module. The command to load the module for "blastp" must also be added to your sbatch script.
3. In addition, your sbatch script should also do the following:
   - run the job in the "normal" queue.
   - give the name "sequence_alignment" to the job.
   - save the standard output to a file named "sequence_alignment.out".
   - save the standard error stream to a file named "sequence_alignment.err".
   - send a notification email to your mailbox when the job is completed.

4. If you wish you can use the template script `HPC_sbatch_script_template.sh` as starting point for your script. This template file is found at the root of the `HPC_exercises` directory.
5. When you script is ready, submit it to the cluster with `sbatch` and verify that it completes successfully.
   **Reminder:** after submitting a job, the following two slurm commands allow you to track your job's progress:
   - `squeue -u $USER`: list currently running or pending jobs.
   - `sacct`: list jobs in running, pending, completed and failed states for the current calendar day.

**Questions:**

- Show your complete sbatch script.
- What output do you get in "blastRun.out"?
- Is the file "blastRun.err" empty? explain why?

---

---

## Exercise 6: requesting additional CPUs and memory

**Objective:** learn how to request additional CPUs and memory for a slurm job, and how to find-out a job's actual memory and CPU resource usage.

**Tasks:**

1. Change directory into `../exercise_6`, where you should find a script named "multithreaded_job.sh". There are also two other R scripts in the directory, but you don't need to worry about them.
2. Open the `multithreaded_job.sh` script with a text editor (e.g. `vim` or `nano`) and set the correct value for the `--account` and `--mail-user` options. Notice that the resource requirements for the job have already been set to 4 CPUs (`--cpus-per-task` option), but no specific request was made for memory (which means that the job will get the default 2GB per CPU).
3. Submit the script to the cluster using the `sbatch` command. After submitting the script, you can track its progress using the `squeue -u $USER` and `sacct` commands.

   **Questions:**

   - did the job complete successfully?
   - If not, what type of problem(s) did you encounter?

4. If the job did not complete successfully, fix the problem by requesting more resources as needed (CPU and/or memory). Then submit your script to the cluster again.

   **Question:** how much resources (CPU and memory) did you request to run your job successfully?

5. Inspect the resource usage of the job you just completed with the following command. Verify whether you are really using all the CPU and memory that you requested:
   `sacct --format=jobid,jobname%15,partition,account%30,alloctres%40,cputime,totalcpu,maxrss,submit,state%15,exitcode,elapsed`

   **Question:** Are the resources requested for your job adequate, or is it possible to make them fit better to the actual requirements of the job?

6. If needed, adapt the resource requirements of your job and run it again.

---

---

# Exercise 7: job arrays

**Objective:** learn how to submit job arrays to the cluster.

**Tasks:** In this exercise we will try to identify all prime numbers between 1 and 50'000. To speed-up things, we decided to divide the 50'000 numbers into 10 subsets of 5'000 numbers. Each subset is stored in a different input file ("sample_1.txt", "sample_2.txt", ...) and must be processed by the "filter_prime_numbers.R" script in order to create an output that only contains prime numbers.

1. Change directory into `../exercise_7`
2. In the directory, you should find a script named `array_job.sh`, as well as the 10 different sample files named `sample_1.txt`, `sample_2.txt`, ... etc.
   To make your life easier, the "array_job.sh" script already contains a good number of slurm options and instructions to run the pipeline, but you still have to do a number of important edits.
3. Edit the slurm options of the "array_job.sh" script and add the correct options to:
   - create a job array with 10 replicates that have indices from 1-10.
   - give the name "arrayTest" to the job.
   - run the job in the "normal" queue.
   - Request 2 CPUs and 2 GB or memory *for each replicate* of the array.
   - set a time limit of 10 minutes *for each replicate* of the array.
   - set the correct values for the `--account` and `--mail-user` values.
4. Edit the "# Job commands." section of the "array_job.sh" script, so that `<input file to process>` is set to the correct input sample file for each replicate of the array.
   **hint:** you might need to use the `$SLURM_ARRAY_TASK_ID` environment variable
5. When your "array_job.sh" script is ready, submit it to the cluster using the `sbatch` command.

---

Last modified: Friday, 20 September 2019, 8:47 AM