EPFL

Parallel and High Performance Computing

Dr. Pablo Antolín

Solution Series 1

February 27 2025

Development and execution in an HPC environment

2 Remote access and file transfer

Exercise 1: ***Simple connection***

To connect to the front node of a cluster if it is your first time.

- `ssh <username>@<machine>`
- `ssh -l <username> <machine>`

Front nodes:

- `helvetios.hpc.epfl.ch`
- `izar.hpc.epfl.ch`

- Connect to `helvetios` front node.
- Check the different folders `/home`, `/scratch`.

Exercise 2: ***Using scp***

How to use `scp` / `pscp.exe`

- Send data to remote machine:
  `scp [-r] <local path> <username>@<remote>:<remote path>`
- Retrieve data from remote machine:
  `scp [-r] <username>@<remote>:<remote path> <local path>`

- Copy a file from your machine to the cluster.
- Retrieve a file from the cluster to your machine.

Exercise 3: ***Connection using keys***

- Generate a pair of public/private key using `ssh-keygen`.
- Copy the public key `.ssh/id_rsa.pub` using `scp` in a file called `.ssh/authorized_keys` on the remote machine.
- Try to connect (it should not ask your password).

3 Compiling code on clusters

Exercise 4: ***Compilation***

GCC: `g++ -o <executable> <sources>`
Intel: `icpc -o <executable> <sources>`

- Compile the code `hello.c`.
- Note: remember to load either `gcc` or `intel` modules, respectively, before invoking the compilations above.

Exercise 5: ***Separated compilation***

`g++ -c <source>`
`g++ -o <executable> <object 1> ... <object n>`

- Enter the folder `hello-separated-files`
- Generate `hello.o`
- Generate `greetings.o`
- Generate `hello` from `hello.o` `greetings.o`

Exercise 6: ***Module command***

- List all available modules
- Try `g++ --version`
- Load the module `gcc`
- Try again `g++ --version`
- Try `icpc --version`
- Load the `intel` module
- Try again `icpc --version`
- List the currently loaded modules

Exercise 7: ***Makefiles***

- Enter the folder `hello-makefile`
- Read the `Makefile`
- Add the `-Wall -Werror` option to the compilation options
- Compile the code
- Introduce an error or a warning in the code (e.g., declare a non used variable).
- Compile the code

4 Submitting a job

Exercise 8: ***SLURM: first commands***

- Check the queue state
- Use `salloc` to allocate one node with `--qos=math-454 --account=math-454`
- Try `srun hostname`, we will see this command more in detail in later exercise sessions
- Exit the allocation to note block resources: `exit` or Ctrl-d

Exercise 9: ***SLURM: command sbatch***

- Write a script that runs the hello world code
- Try your script.
  note*: in general you should not try your codes on the front node there is a* ***debug*** *partition for that*
- Submit your script with `sbatch` using the QOS
- Try `squeue -u <username>` or `Squeue`
- A file named `slurm-<jobid>.out` should have been created, check its content Add the QOS as an option directly in the script
- Submit it again

5 Basics on GIT

Exercise 10: ***First step with Git***

- If you do not have git installed, get it from https://git-scm.com/downloads or from your package manager
- Go on https://gitlab.epfl.ch and login with your EPFL credentials.
- Once connected go to the preferences page (left bar, user icon on top)
- In the **User settings** > **SSH Keys** menu add your public ssh key. This key will be used to connect to the git server through **ssh**.

Exercise 11: ***First steps with Git***

- `git clone <repo url> [local name]`: Clone a repository
- `git add <files...>`: Stage modified/new files
- `git commit -m "comment"`: Commit staged files
- `git pull`: Pull and merge remote modifications
- `git push`: Push the local modifications to the remote server
- `git status`: Check the local state

- Now you should be able to clone a repository
  Either create a repository or `git clone git@gitlab.epfl.ch:math454-phpc/test-repo.git`
- Create a file, use a filename that will not clash with the others
- Check the state of your working copy
- Add the file to the repository
- Commit your modifications
- Clone the same repository in a different folder
- Pull the potential modifications from the server
- Push your changes to the server

Exercise 12: ***Generate and solve conflicts***

- Modify the file created in the previous exercise in both clones
- Commit this both modifications

- Pull and push in one of the clone
- Pull in the second clone, You should get a conflict

```
<<<<<<<<<<
One version
==========
Other version
>>>>>>>>>>
```

- Check the local status
- Correct the conflict and commit using `git commit -a`
- Push the modifications

Exercise 13: ***Branches / merges***

- `git branch <name>`: Create a new branch from the current HEAD
- `git checkout <name>`: Switch to the specified branch
- `git merge <name>`: Merge the branch specified in the current one
- `git branch -d`: Delete a branch
- `git branch -a`: List all branches
- `git log`: List the different commits of the current branch
- `git log --graph --all`: Show also the branches

- Create a branch with the name of your choice
- Modify a file and commit the changes
- Checkout the `master` branch
- Modify a file and commit the changes
- Merge the branch previously created in the `master` branch
- List all branches
- Print the logs of the different modifications
- Delete the merged branch

Exercise 14: ***Handle remotes***

- `git init --bare`: Create a new server
- `git remote add server <url>`: Add a remote server
- `git remote -v`: Show the remotes configured
- `git push <remote name>`: Push to a given remote

- Connect on the front node of your favorite cluster
- Create a new folder that will contain your server
- In this folder initialize a new git server
- In one of the former clone of `scitas-test` add the new remote URL `<cluster name>:<path to repo>`
- List the remotes to see if everything looks correct
- Push the local content to the new server
- On the cluster clone this new server URL `<path to repo>`
- Note: The access permission on this new server are based on the file system permissions