## Exercise Introduction

**PLEASE COMPLETE TASK 1 AND TASK 2 BEFORE THE BEGINNING OF THE COURSE ON MONDAY!**
Task 1 gives instructions for connecting to the GWDG systems via SSH.
Task 2 explains how to set up X2Go to use a simple desktop environment via SSH and xterm.
Task 3 is optional and can be completed to set up a Linux environment on your home machine.

## Course Format: Practicals

The Practical Course High-Performance Computing 2022 takes place in an online format utilizing two Big Blue Button rooms.
The main room is called **Broadcast - PCHPC**. In this room the lecturer will present the slides and guide you through the course. The PCHPC features a wide set of lecturers from the GWDG and the university, who are experts for the topics they are presenting.
As this course is intended to provide hands-on experience, the lecturers will ask you to complete exercises during the course. These exercises should be completed individually, however, you will form groups to support each other in case you get stuck. To allow for communication within said groups, each group will receive its own breakout room in BBB. The second BBB room called **Breakout - PCHPC** will be used for this. We will use two BBB rooms as Big Blue Button is limited and it is currently not possible to be connected to a breakout room while also being able to listen the main room. If you need help from outside your group, feel free to ask for help in the broadcast room where the lecturer and a few helper will be available. The format will be explained in more detail during the first session.

**For the beginning of the course it is enough to join the Broadcast room.**

**Broadcast Room:** `https://meet.gwdg.de/b/jul-tff-dq1-rt0`

**Breakout Room:** `https://meet.gwdg.de/b/jul-ffv-ljs-7u5`

**Please confirm before the course that can connect to a BBB room and your microphone is working.** You may use the breakout room BBB instance to test your setup[1]

## Contents



[1] If it doesn't work, please try first `https://test.bigbluebutton.org/`, then try with a different browser.

## Task 1: SSH setup and Connecting to the GWDG machines (0 min)

In order to follow along with the hands on exercises of this course, you need to login to the GWDG Scientific Compute Cluster (SCC). If you have signed up for the course[2], you should have received an SSH key per e-mail, which can be used to login into the SCC.

The following instructions will help you connect to the SCC using your SSH key:

**Windows 10/11:**

- Search for **Powershell**, right click, run as administrator
- `Get-WindowsCapability -Online|Where-Object Name -like '*SSH*'`
  If SSH client is not installed run the following command:
  `Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0`
- Confirm that it works by running `ssh -V`

**MacOS/Linux:**

- Search for **Terminal** and open it
- Check ssh is provided by running the command `ssh -V`

**Using SSH:**

- Place the SSH key you received per mail in your user folder
- In PowerShell or Terminal type the following command
  ```
  ssh -i hpctrainingNN hpctrainingNN@login-mdc.hpc.gwdg.de
   -o ProxyCommand='ssh -W %h:%p hpctrainingNN@login.gwdg.de
   -i hpctrainingNN'
  ```
- Confirm the connection and enter the SSH keys passphrase **twice**
- The passphrase is described on the exercise sheet you received
- Confirm that running `hostname` returns *gwdu101* or *gwdu102*

This SSH command connects you as user *hpctrainingNN* to the SCC using the SSH key file with the same name. The proxy command is used to connect to the SCC over login.gwdg.de. This is necessary as the SCC is only reachable from inside the GÖNET. If you are already connected via the GWDG VPN or from a device inside the GÖNET, you do not need the proxy command.

### Hints

- If you get an error stating that the permissions of your ssh key are too open, you have to limit the files permission. Type `chmod 400 hpctrainingNN` to fix the permission.
- If you get an error stating that the format of your key is invalid, try opening the key file with a text editor and make sure it starts with `-----BEGIN OPENSSH PRIVATE KEY-----` and ends with `-----END OPENSSH PRIVATE KEY-----`. You can also try copying the content into a new file. Make sure that there is an empty line at the end of the file.

[2] If you signed up late, you might not have received a key.

## Task 2: Graphical Interface via X2Go and xterm (0 min)

A few of the exercises utilize programs with a graphical user interface. We utilize X2Go to access a simple remote desktop running on the SCC via SSH.

The following instructions will explain how to setup X2Go on your machine:

**Setup:**

1. Download and install the X2Go client for your operating system: `https://wiki.x2go.org/doku.php`
2. Start X2Go and create a new session if it does not prompt you to create one
3. Set **Session name** to a name such as `PCHPC 2022`
4. Set **Host** to `login-mdc.hpc.gwdg.de`
5. Set **Login** to `hpctrainingNN` , where **NN** is the number in the key file name
6. Set **Use RSA/DSA key for ssh connection** to the ssh key file you received from us
7. Tick **Use Proxy server for SSH connection**, which should open another menu
8. Set **Type** to **SSH**
9. Set **Host** to `login.gwdg.de`
10. Set **Login** to `hpctrainingNN`
11. Set **RSA/DSA key** to the ssh key
12. Set **Session type** to **Single application**
13. In the new drop down menu on the bottom right select **Terminal**
14. Switch to the third tab **Input/Output** and set **Display** to **Custom** and a resolution you are comfortable with
15. Confirm and close the settings page

**Connecting:**

1. Open X2Go if it is not already open
2. Double click on the session you created for this course or type in its name and press ENTER
3. Enter the passphrase of your ssh key `hpctrainingNN-XXX` **twice**
4. After a delay you will be connected to the SCC in an xterm session

You can confirm that it works by running `module load vampir` and `vampir` , which should open vampir in another window.

**MacOS** users might need to install XQuartz as prompted by a dialog.

## Optional Task 3: Setting up a local Linux environment and Bash shell (0 min)

This is a difficult **additional** task that will support your understanding in the topic.

This is an optional task, no extra time is allocated for it during the live session. Here we present five variants for running a Linux environment on your personal computer. Depending on your hardware, all variants or none

of them may work. All exercises as well as the rest of the course can be completed via the provided GWDG machines.

Variant 1 consists of running a virtual machine (VM) on your computer. Your host system needs to have enough resources available to effectively simulate a second operating system. These resources are than reserved by the VM such that they are no longer available to the host. Your system should have at least 2 GB RAM (recommended 4 GB), 2 CPU cores (recommended 4 cores) and at least 8 GB of disk space (recommended 20 GB) to spare.

Variant 2 utilizes the Windows Subsystem for Linux (WSL) introduced in Windows 10. This installs a Linux system, which is managed through Windows and can share resources with Windows such that no reservation is necessary.

Variant 3; MacOS is based on the same type of operating systems as Linux such that it can run Bash natively. In this variant, the default shell is set to Bash and updated.

Variant 4 gives yet another approach by using a type of lightweight VMs called containers to run a Linux system. This involves installing Docker on your host system.

Variant 5; Linux dual boot. This approach should be mentioned but will not be discussed in detail. The idea is to set up a second operating system on your home computer that runs a Linux distribution. Depending on your hardware and available storage space, a dual boot setup might not even be an option for you. Furthermore, it requires partitioning your hard drive to make space for a second operating system, which can lead to the deletion of the data on your home system and even prevent your home system from booting at all if done incorrectly. Only do this if you know what you are doing. Nevertheless, this approach offers the most authentic Linux experience besides using a computer that only runs Linux.

## Variant 1 Linux VM via VirtualBox

Check if your system supports hardware virtualization:

1. On Windows press WIN + r
2. Type in `cmd` and press enter
3. Type in *systeminfo* and press enter
4. After a moment it should show a message about the status of **Hyper-V** at the bottom
5. If its enabled you are done, if its not you need to enable it, if its not supported, you cannot use VirtualBox

Enabling virtualization:

1. Reboot your system into the BIOS by pressing the BIOS key for your system during boot, commonly its one of these: Esc, Del, F1, F2, F4, F12
2. In the BIOS navigate to CPU settings and look for a virtualization setting and enable it
3. Save the changes and reboot
4. On Windows, search for a menu called **Turn Windows features on or off** and open it
5. Search for a feature called **Hyper-V**, enable it, press okay and wait for it to install it
6. Reboot again

Installing VirtualBox and VM setup:

1. Download and install VirtualBox on your system `https://www.virtualbox.org/wiki/Downloads`
2. Download a Linux VM image, to skip the install process and get started right away, we recommend a pre-installed image from OSBoxes.org. Download the newest 64-bit Ubuntu image `https://www.osboxes.org/ubuntu/` and check default username and password under info. Its commonly *osboxes* as username and *osboxes.org* as password.

3. Open VirtualBox and press **New**
4. Set a name for it and a folder where the files can live
5. Set type as Linux and version as Ubuntu 64-bit and press next
6. Set memory size to 2048 MB or at least 1024 MB depending on your hardware and press next
7. For hard disk selection press **Use an existing hard disk file** and open the selector
8. In the selector, add the **.vdi** file you downloaded from osboxes.org, confirm and press create
9. Now you can start the VM via VirtualBox and you should shortly see a login prompt
10. Use the default username and password to login
11. Now you have Ubuntu as a Linux OS running in virtualization on your machine. Search for terminal or press CTRL+ALT+t to open a shell
12. Run `sudo apt update && sudo apt upgrade -y` to get all updates

## Variant 2 WSL (Windows only)

To be able to use WSL, you must have at least Windows 10 64-bit Build 18917. Check your version by pressing WIN+r, entering `winver` and pressing ENTER.

Next you need to enable the WSL Windows feature.

1. Search for a menu called **Turn Windows features on or off** and open it.
   Alternatively press WIN+r and type in `OptionalFeatures` (capital "O" and "F") and press ENTER
2. Search for a feature called **Windows Subsystem for Linux**, enable it, press okay and wait for it to install it
3. Reboot your system

Set WSL to use WSL 2. Open a PowerShell console and type `wsl --set-default-version 2` . To see your installed subsystem type `wsl -l -v` .

Now you can pick a Linux distribution to install. Open the Microsoft store and search for **Ubuntu LTS**, pick the newest version and press get and after the download open. After installing you will be prompted to set the a user name and password for Ubuntu.

If everything is set up correctly, you can run `wsl` in a PowerShell or command prompt and you should end up with a ready to use Bash shell. Run `sudo apt update && sudo apt upgrade -y` to get all updates.

## Further Reading

- https://adamtheautomator.com/windows-subsystem-for-linux/

## Variant 3 Terminal Bash (MacOS only)

MacOS already comes with either Bash or Zsh as the default shell. These can be used to follow along with most commands. However, some commands may work differently or not at all via this shell as MacOS does not run a fully featured Linux kernel.

Setting Bash as the default shell:

1. Search for Terminal application and open it
2. Run `echo "$SHELL"` to see what shell is set as default, if its **/bin/bash**, you are done
3. Else, you need to change it to Bash. List your available shells using `cat /etc/shells`

4. If **/bin/bash** is in there you can make it your default shell using `chsh -s /bin/bash`
5. Close the terminal and open it again, type `echo "$SHELL"` to confirm it worked

The default Bash installation might be severely outdated.

1. To check your Bash version type `bash --version` , if the version is outdated, follow these steps to upgrade it
2. Go on `https://brew.sh/` and install it by pasting the command into the terminal
3. Verify the installation by running `which -a bash` , this should list two bash instances, one in **/bin/bash** and one in **/usr/local/bin/bash** or similar
4. Verify the version of the installed bash binary via `/usr/local/bin/bash --version` and compare it to `/bin/bash --version`
5. Add the new shell to the list of shells via `sudo nano /etc/shells` and add in a new line `/usr/local/bin/bash` and save
6. Set the default shell as the new bash shell for yourself `chsh -s /urs/local/bin/bash` and for the admin user `sudo chsh -s /urs/local/bin/bash`
7. Reopen terminal and `echo $BASH_VERSION` as well as `sudo echo $BASH_VERSION` should give the new Bash version

### Variant 4 Docker container

On Windows, Docker utilizes the WSL system for running container, so if possible, you should use the WSL directly. This option also enables MacOS hosts run a regular Linux environment.

1. Go to `https://www.docker.com/products/docker-desktop/`, download and install Docker desktop for your system
2. Follow the post-installation instructions from Docker desktop to complete its setup
3. Open Terminal and run `docker run --name ubuntu -it -v $(pwd):/tmp ubuntu /bin/bash` or a PowerShell and run `docker run --name ubuntu -it -v ${pwd}:/tmp ubuntu /bin/bash`
4. This creates a container running Ubuntu, mounts your local directory under **/tmp** in the container and gives you a bash shell
5. To close the container type `exit` , to restart it later on use `docker run -it ubuntu /bin/bash`

The container image uses a slimmed down version of Ubuntu such that many command tools might not be installed by default and need to be added retrospectively. Inside the container you operate as a super-user by default and **sudo** is not installed, so you can `apt update && apt upgrade -y` without the prefix.

On Windows, if you have previously enabled the **Hyper-V** Windows feature, you might need to disable it before being able to run Docker containers. You do not need to disable virtualization in the BIOS, only the Windows **Hyper-V** feature.