Department of Psychology
Royal Holloway, University of London
Egham
Surrey
TW20 0EX

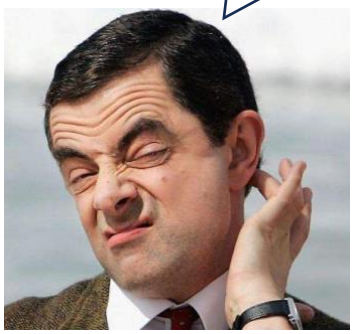
Psychp01 Manual

Gabriele Bellucci

v. 1
2024

To know everything!

To get it done!



Choose what you want to do
by clicking the corresponding text in the speech balloons above

Table of Contents

Prerequisites

Psychp01 is a High Performance Computing (HPC) cluster as a vitualized system that runs Debian 11 (https://www.debian.org/), a Linux operating system. You will need to become familiar with the Linux command line interface to use them effectively. While it can be time consuming to learn a bit of Linux it can be considered an investment in scientific skills.

This YouTube.com video starts from the very basics of the Linux command line. And Fig.1 shows a nice cheat sheet for Linux.

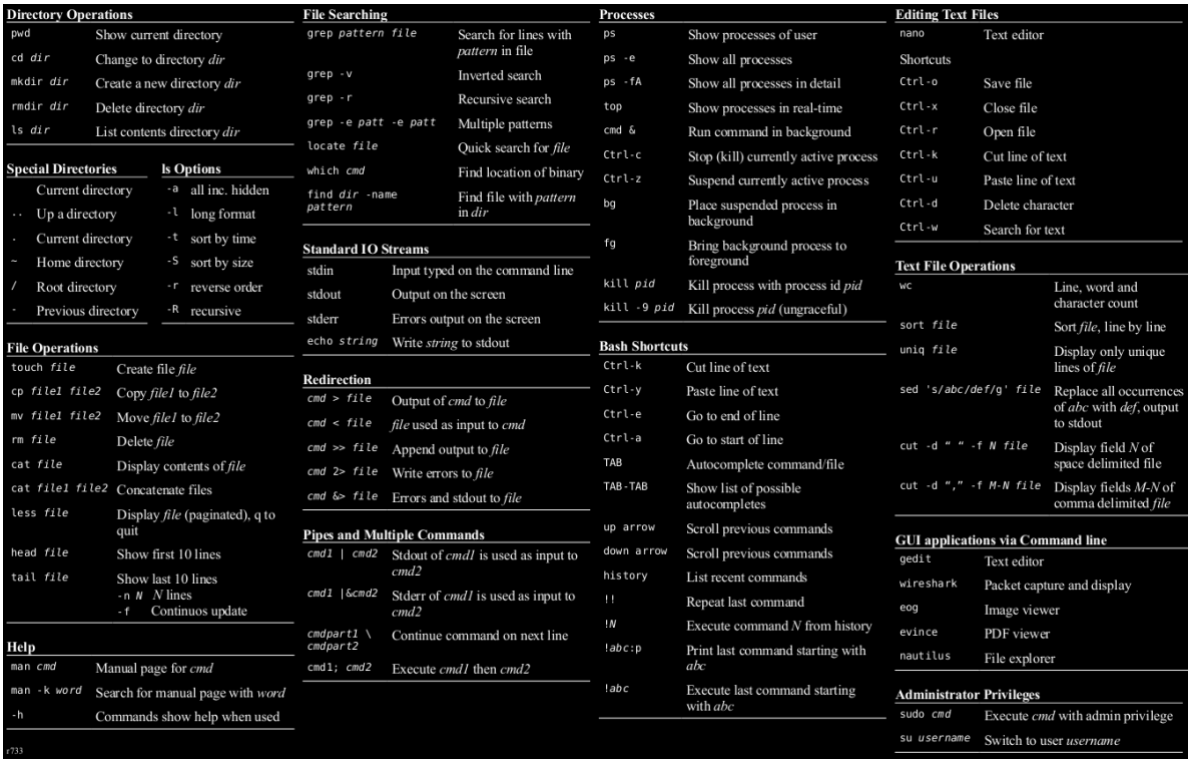


Figure 1. A nice cheat sheet for Linux

Also, please find here some useful tutorials on HPC clusters. Moreover, the department of Psychology has a github page with tutorials and resources to help you get started, guide you through some practicalities, and provide you with useful code snippets.

Moreover, for any technical assistance, please send a ticket to the IT help desk at itservicedesk@rhul.ac.uk, or email Gabriele Bellucci at gabriele.bellucci@rhul.ac.uk.

Getting Started with HPC on psychp01

Objective: Here you will get to know psychp01 and will learn how to connect to psychp01.

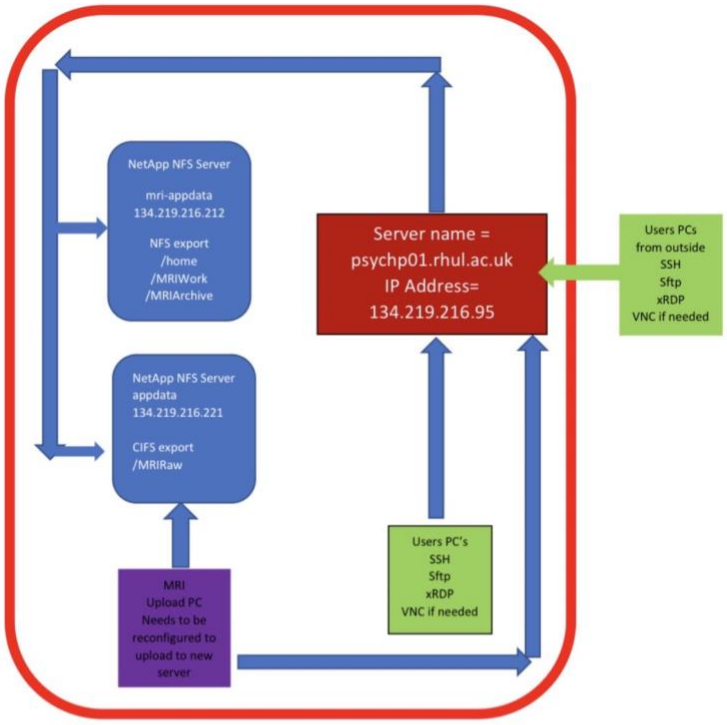
What's psychp01?

Psychp01 is a virtual computer cluster in a cloud environment at Royal Holloway. Psychp01 provides local HPC resource with an end user experience similar to most HPC Linux clusters.

Current available resources are 128 cores and 256 GB memory. The server mounts NFS filesystems from a NetApp File Server (see Fig. 2).

Psychp01 utilizes Linux (Debian), a batch scheduler (SLURM), and various software packages deployed using a module system.

There will be an option for Singularity modules which enable applications and user to bring their own software environment and preserve that environment in the name of reproducible research.



Red outer line indicates VPN. Users from home are given firewall access so that they can connect to the server using SSH, SFTP, xRDP and VNC protocols.

The cluster has 128 cores (4 times as much as we had before) and 256 GB memory (compared to 192 GB previously). This has been at no extra cost and I am hoping to double this at some point soon, again hopefully at no cost.

Psychology server **psychp01.rhul.ac.uk** mounts NFS filesystems from a NetApp File Server. The decision has been taken to install applications locally on the server

Figure 2. Psychology cluster specifications.

First Steps on psychp01

Overview

It is strongly advised that new HPC users explore the many tutorials and documentation resources available on the web, for example: HPC Cluster Tutorials or our RHULPsychology github page Please see Prerequisites.

Before starting, you need to have a short introduction to the usage of the cluster and current guidelines in place at the department. For that, please contact Gabriele Bellucci at gabriele.bellucci@rhul.ac.uk.

On psychp01, there are four main locations you'd need to get familiar with:

- /home
- /MRIWork
- /MRIArchive
- /MRIRaw

We will explain to you step by step how to move to these locations and what they are there for. Importantly, data in the /MRI* locations will be backed up to avoid data loss.

See also this CUBIC wiki page for further details on data structure, storage, backup and data sharing.

Account and Password

Access to psychp01 can be made available to all staff members with a @rhul.ac.uk email address. To request for an account for HPC access, please send an email to itservicedesk@rhul.ac.uk. It is expected to take 2-3 days for creating your account and corresponding access to the system. Someone from the Psychology IT team will get in touch with you shortly.

When you get confirmation, you will be able to connect to the cluster.

Access to psychp01

Access to psychp01 can be performed through the command line (for computing purposes) as well as the Graphical User Interface (GUI; for visualization purposes only). Here, the ways of access to the cluster are described.

Psychp01 Environment

Once you have logged into psychp01, you are in a basic Linux Debian command line environment. You will need to be familiar with the basics of the Linux command line interface to use psychp01. Luckily, there are many good tutorials on the web to help with this.

On psychp01 you can setup compute jobs and submit them for processing. You can have an interactive environment enabling you to edit files, write scripts, load software modules and compile programs. You can download resources from the internet such as git repositories or singularity containers.

Psychp01 is a batch computing system which means you must submit your computational work to a job scheduler, in our case SLURM. To submit a job to the scheduler you will need to create a job script. Creating job script is so key to batch HPC cluster computing that if you are not familiar with batch jobs and SLURM, please see the Running Jobs section of this document.

Files and Data Access

When you `ssh` login to psychp01, you'll be in your cluster home directory

```
/home/username
```

There will be a quota on this directory of 1.5TB. This is different from your campus home directory or network file share. This is a place where you can setup your programs and scripting for jobs that will be submitted to run on the compute resources.

Given the small space of your home directory, no data should be uploaded to it. Instead, you should place all your data into `/MRIWork`, which is your workspace. A specific workplace folder will be created and provided by the psychology IT team, which is regularly backed up. It is encouraged that the users keep the workspace clean and all cached and unnecessary files to be deleted from the workspace at regular intervals.

Data Staging

To move files from your computer to psychp01 or vice versa, you may use any tool that works with `ssh`.

On Linux and OSX, these are `scp`, `sftp`, `rsync`, or similar programs. Please see Transferring files to/from psychp01.

On Windows, you may use VNC.

Linux QuickStart

New on Linux systems?

This page contains some tips on how to get started using the psychp01 cluster if you are not too familiar with Linux/Unix. The information is intended for both users that are new to psychp01 and for users that are new to Linux/UNIX-like operating systems. Please consult the rest of the user guide for information that is not covered in this chapter.

Please also visit this MIT course on basics shell commands and command line environment.

SSH

On a terminal, you can use `ssh` to login to psychp01. Check the Access to psychp01 through Command Line in this documentation to learn more about `ssh`.

File Transfer Clients

To transfer any file or data you wish to use to the cluster, you can use file transfer clients, such as `scp` or `sftp`. For more info, see Transferring files to/from psychp01.

Running jobs on the cluster

You must execute your jobs by submitting them to the batch system using bash files. There is a dedicated section on bash files and batch system in our user guide that explain how to use the batch system. The pages explain how to write job scripts, and how to submit, manage, and monitor jobs. It is not allowed to run long or large memory jobs interactively (i.e., directly from the command line).

Common Commands

Provide the full path name of the directory you are currently in:

```
pwd
```

Change the current working directory:

```
cd
```

List the files and directories which are located in the directory you are currently in:

```
ls
```

Searches through one or more directory trees of a file system, locates files based on some user-specified criteria and applies a user-specified action on each matched file:

```
find
```

Find specific files, you can use the -name and -type arguments:

```
find . -name 'my*' -type f
```

The above command searches in the current directory (.) and below it, for files and directories with names starting with my. “-type f” limits the results of the above search to only regular files, therefore excluding directories, special files, pipes, symbolic links, etc. my* is enclosed in single quotes (apostrophes) as otherwise the shell would replace it with the list of files in the current directory starting with “my”.

Find a certain expression in one or more files:

```
grep
```

For example, to find the string `apple` in the `fruitlist.txt` text file, type:

```
grep apple fruitlist.txt
```

Create new directory:

```
mkdir
```

Remove a file. Use with caution:

```
rm
```

Remove a directory. Use with caution:

```
rmdir
```

Move or rename a file or directory:

```
mv
```

Edit text files in command line (for more details, see here):

```
vi
```

or

```
vim
```

View (but do not change) the contents of a text file one screen at a time, or, when combined with other commands (see here) view the result of the command one screen at a time. Useful if a command prints several screens of information on your screen so quickly, that you don’t manage to read the first lines before they are gone.

```
less
```

and

```
more
```

Use the “pipe” or “vertical bar” to group two or more commands together:

```
|
```

For example, list files in the current directory (ls), retain only the lines of ls output containing the string ``key” (grep), and view the result in a scrolling page (less), type:

```
ls -l | grep key | less
```

A complete listing of the defined variables and their meanings can be obtained by typing:

```
printenv
```

You can define (and redefine) your own variables by typing:

```
export VARIABLE=VALUE
```

If you know the UNIX-command that you would like to use but not the exact syntax, consult the manual pages on the system to get a brief overview. Use `man command` for this. For example, to get the right options to display the contents of a directory, use:

```
man ls
```

To choose the desired options for showing the current status of processes, use

```
man ps
```

Text editing in command line

A popular tool for editing files (e.g., your code) on Linux/UNIX-based systems is `vi`. Unfortunately, the commands within both `vi` editor are quite cryptic for beginners. It is probably wise to spend some time understanding the basic editing commands before starting to program the machine.

For quick help, use:

```
man vi
```

or

```
man vim
```

Suppose you want to check the progress of your analyses. Suppose that you defined in your bash file that SLURM is supposed to output the status of your analyses in a text file (see here) named `job_out.txt` stored in your `analyses` folder in your home directory `/home/gbellucci`. To quickly check the status of your analyses, you would first establish an `ssh` connection with the cluster, then `cd` to your `analyses` folder and type `vim job_out.txt`, like that:

```
ssh gbellucci@psychp01.rhul.ac.uk
gbellucci@psychp01.rhul.ac.uk's password:
gbellucci@psychp01:~$ cd /home/gbellucci/analyses
gbellucci@psychp01:~$ vim job_out.txt
```

This would open the file in your command window. You can scroll and check the status of your analyses. To exit just type:

```
:q
```

Every file opened with `vim` will be in a non-writable state. To write in your opened file, press

```
i
```

On the bottom left of your command line, you will see the word `INSERT` appear, like that:

```
--INSERT--
```

Modify your file as you wish. When done, press `ESC` on your keyboard. This will make the word `INSERT` disappear. You are again back to a non-writable state. To save and exit, type:

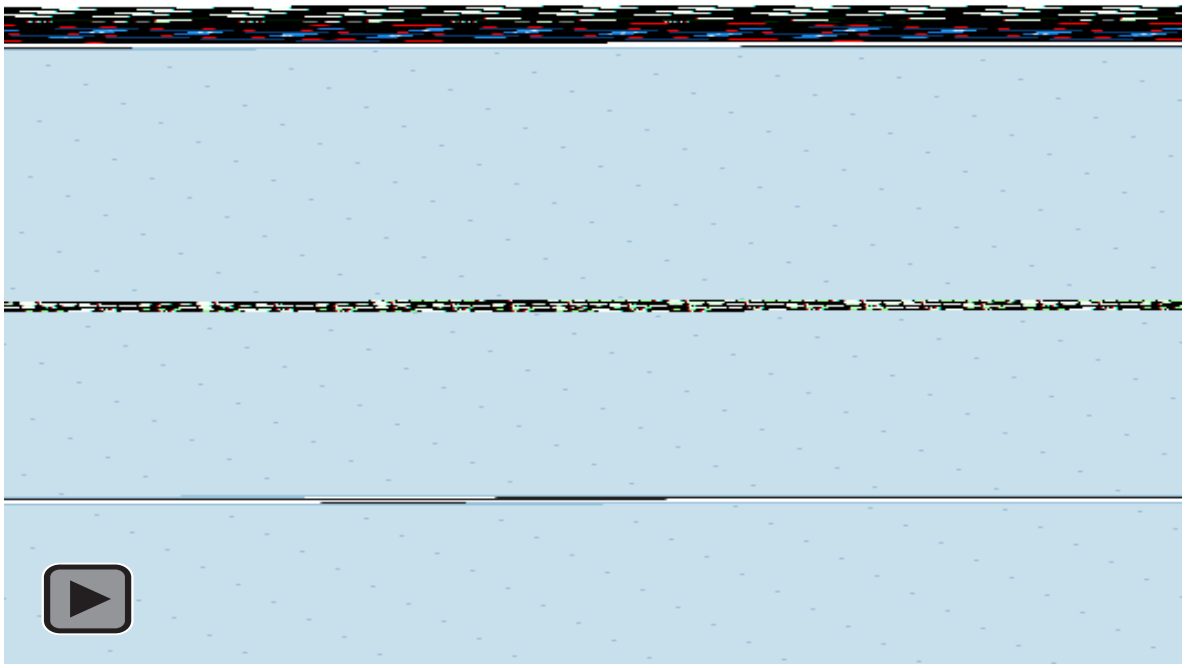
```
:wq
```

Where `w` stands for “write” and `q` for “quit”. If you change your mind and decide to exit without saving your changes, just type

```
:q!
```

This will not write your changes and force an exit (that why the `!`) from `vim`.

Please see below a short video demonstrating that.



Cluster access

After you have registered up for HPC access, you will be provided with a local USERID and PASSWORD, different from your college login credentials, which you can use to log into psychp01 using `ssh` (secure shell).

To access the cluster, you need to be in the campus network.

If you are connecting form a Windows computer on campus, you are already in the campus network.

If you are on campus and connecting from a Linux or Apple computer or if you are off campus, you will need VPN access to the campus network and then connect to psychp01.

If you don't have VPN access, please submit a ticket to the IT help desk.

VPN connection

Windows PC:

On campus:

1. Establish an `ssh` connection

Off campus:

1. Connect to VPN
2. Establish an `ssh` connection

Mac or Linux

Both on and off campus:

1. Connect to VPN
2. Establish an `ssh` connection

For Windows, you will need to have installed an SSH client.

Finally, there is also a GUI option, which, however, is not for computing purposes.

See here for a step-by-step guide to install the VPN client GlobalProtect.

Log in with `SSH`

Access to psychp01 through Command Line (Linux and OSX)

An `ssh` client (Secure SHell) allows you to connect to psychp01 from your terminal. An `ssh` client provides secure encrypted communications between two hosts over an insecure network.

If you already have `ssh` installed on your UNIX-like system (Linux or Apple OSX), have a user account and password, login may be as easy as opening a terminal and typing `ssh` *cluster_name*, for instance:

```
ssh psychp01.rhul.ac.uk
```

into a terminal window.

If your username on the cluster differs from your username on the local machine, use the -l option to specify the username on the machine to which you connect. For example:

```
ssh cluster_name -l your_cluster_username
```

or alternatively,

```
ssh your_cluster_username@psychp01.rhul.ac.uk
```

Hence, if my cluster username is "gbellucci", I would type the following to access the cluster:

```
ssh gbellucci@psychp01.rhul.ac.uk
```

If you need X-forwarding, you must log in like this:

```
ssh -X cluster_name
```

For Interactive sessions, for example MATLAB, you should add -Y to your `ssh` command:

```
ssh -Y username@psychp01.rhul.ac.uk
```

where the command -Y enables trusted X11 forwarding.

Once you have hit entered, you will be asked to provide your *password*. Once you entered your password, you will be connected to the cluster. You will see some basics information about the cluster printed out in the terminal like:

```
Linux psychp01 5.10.0-24-amd64 #1 SMP Debian 5.10.179-5 (2023-08-08) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/ */copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Oct 6 19:08:37 2023 from 10.40.65.14
```

At the beginning of your terminal line, you will see your username and the server name like that:

```
username@psychp01:~\$
```

When you are done, you can close the connection just by typing `exit` in the command line. Please see below a short video that demonstrates that.

SSH clients for Windows

On Windows, you can use PuTTYgen that comes with PuTTY. More information on ssh.com. Further, the cmder console emulator works nicely with Windows 10/11 and is available open source here. Please download the full version to get built-in installation of git-for-windows along with it. If you have any queries, please get in touch with the IT.

At the OpenSSH page, you will find several SSH alternatives for both Windows and Mac.

Please note that Mac OS X comes with its own implementation of OpenSSH, so you don't need to install any third-party software to take advantage of the extra security SSH offers. Just open a terminal window and jump in.

To learn more about using SSH, please also consult the OpenSSH page and take a look at the manual page on your system by typing the following on your terminal:

```
man ssh
```

GUI access

Alternatively, for allowing GUI access, the cluster comes installed with Remote Desktop Protocol (RDP). From a Windows and Apple computer, use the Microsoft Remote Desktop App. From Linux computers use Remmina. You should use the server location as psychp01.rhul.ac.uk and use the local userid and password provided by Psychology IT for logging in through RDP. The advantage of RDP is that your desktop will be preserved between logins, across network disconnects, etc, until the cluster is reset.

IMPORTANT: Be aware that the GUI access is a useful option you can use in case you would need to visually inspect something on cluster without the need to download it onto your local machine (e.g., plots of your results or preprocessing steps and so on). However, you should not

run anything from the GUI. All analyses need to be run using the batch system. Processes that do not use the batch system will be killed.

Running analyses on psychp01

To run analyses, you:

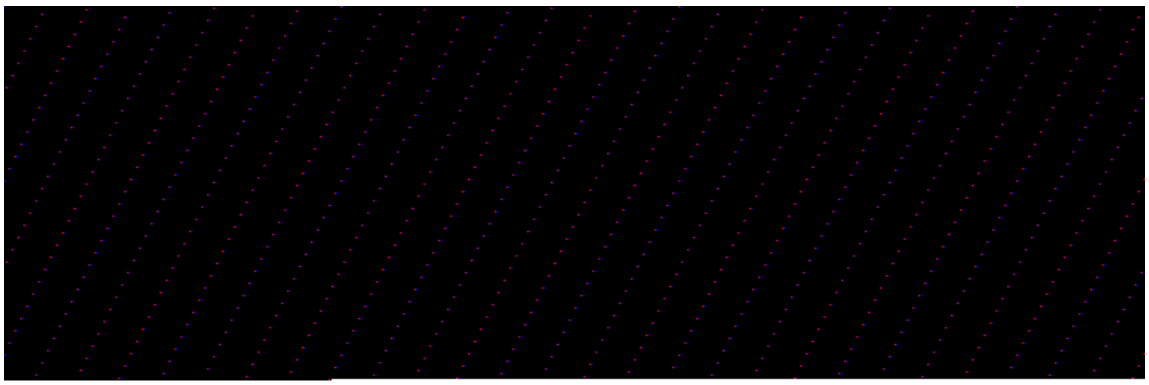
1. **need access to the cluster**.
    a. No access? Go to How can I access the cluster?

2. **need your data and code on the cluster**.
    a. To see how you can transfer them go to How can I transfer files to the cluster?

3. **need your job file on the cluster**.
    a. See How can I set up a job file?

4. can then **run** `sbatch *your_file_name.sh*` **in the command line**.
    a. For more details, see How can I run a job file?

Please see below a short video demonstrating that.



Transferring files to/from psychp01

Transferring data

You need to upload your data into the folder that was created by the IT for you within the `/MRIWork` folder on the cluster. Your folder name will be something like `MRIWork#` where `#` stands for a number assigned to you. Hence, if your number is "25", your path name to upload your data on the cluster will be:

```
/MRIWork/MRIWork25
```

The most secure way to transfer data is using File Transfer Clients like `sftp` and `scp` via the command line.

To know more about /`MRIWork`, folder structure, data archiving, data backup and data sharing, have a look at the CUBIC wiki.

Please NOTE: theoretically, you can upload your data also into your home directory (see **Transferring code** below) and save your analysis results there. However, the home directory has limited amount of storage space and is **not** periodically backed up. Hence, it is advisable that you save your data and analysis results in your `/MRIWork/MRIWork#` folder, even if you do not have MRI data and do not think your data occupy too much space. This will avoid problems with storage space and minimize risks of data loss.

Transferring code

You can upload your scripts into a folder within your home directory on the cluster. The IT will have probably created a folder in the home directory with the first letter of your name preceding your surname (e.g., `gbellucci`). Hence, if your name is Gabriele Bellucci, your path name to upload your data on the cluster will be:

```
/home/gbellucci
```

Your folder name in the home directory has the same name as the username you use to access the cluster. This name was provided to you by the IT when you asked for access. Be aware that there might be deviations on how your folder name in the home directory has been created (especially if you were granted access to the cluster before 2024). If you have access to the cluster, just log in using `ssh` and type `pwd` to see your folder name in the home directory. Something like the line above will pop up on your command line.

The most secure way to transfer code is using File Transfer Clients like `sftp` and `scp` via the command line. Please see below for how to use these clients and a demonstration video.

File Transfer Clients

Primary access to psychp01 is via `ssh` based tools (on the command line). To upload or download data and code, *File Transfer Clients* such as `scp` and `sftp` can be used.

To transfer data to and from psychp01 use the following address:

```
psychp01.rhul.ac.uk
```

sftp

`sftp`, which stands for *Secure File Transfer Protocol*, is an encrypted protocol built into `SSH` that can implement commands for transferring files between two remote systems over a secure connection. There are many resources on the web on how to use `sftp` (e.g., here). Here, example applications to transfer data onto psychp01 will be shown.

First, you need to establish a secure connection with the server. This is very similar to how you would connect with the server using `ssh` (see here).

```
sftp username@psychp01.rhul.ac.uk
```

Like for the `ssh` connection, "username" is the username provided to you by the IT when you asked for access to the cluster. Hit enter and you will be required to enter your password. Once you are connected, at the beginning of your command line, you will see that an connection has been established:

```
sftp>
```

Now, you can `ftp` commands to (among others) upload, download, remove, and move files. Type `help` to check all commands available.

```
sftp> help
```

The `sftp` connection puts you on the cluster. Here, you can use all common commands you would use on your local machine to get the current directory, change the current directory and so on. If you would like to use the same commands on *your local computer*, you can do that by adding an "l" in front of the command you want to use. This "l" stands for "local" and tells `sftp` to use the command on the local machine as opposed to the remote one.

For instance, when you establish an `sftp` connection, you will find yourself in your home directory. Hence, if your home directory path is `/home/gbellucci`, when you type `pwd`, you will see the second line of the code below appearing:

```
sftp> pwd
Remote working directory: /home/gbellucci
```

On the contrary, if you type `lpwd`, you will see the second line of the code below appearing:

```
sftp> lpwd
Local working directory: /Users/Gab
```

where `/Users/Gab` is my (local) current directory on my computer. Type `help` to see the difference in the commands for remote and local implementations.

To download data from the cluster onto your local directory, you need to use the `get` command, like this:

```
sftp> get remote_filename_path local_dirpath
```

For example, if you have to get a file called `results_matrix.mat` from the folder `results` in your home directory `/home/gbellucci` and download it in your folder `project_results` on your local directory `/Users/Gab`, you will do:

```
sftp> get /home/gbellucci/results/results_matrix.mat /Users/Gab/project_results
```

Alternatively, you can `cd` to `results` (on the cluster), `lcd` to `project_results` (on your local machine), and then just type `get results_matrix.mat`, like this:

```
sftp> cd /home/gbellucci/results
sftp> lcd /Users/Gab/project_results
sftp> get results_matrix.mat
```

NOTE: If you have folder names that contain spaces, `sftp` would fail. For instance, something like that: `sftp> lcd /Users/Gab/project results`, (i.e., your results folder named "project result" with a space) would not work!

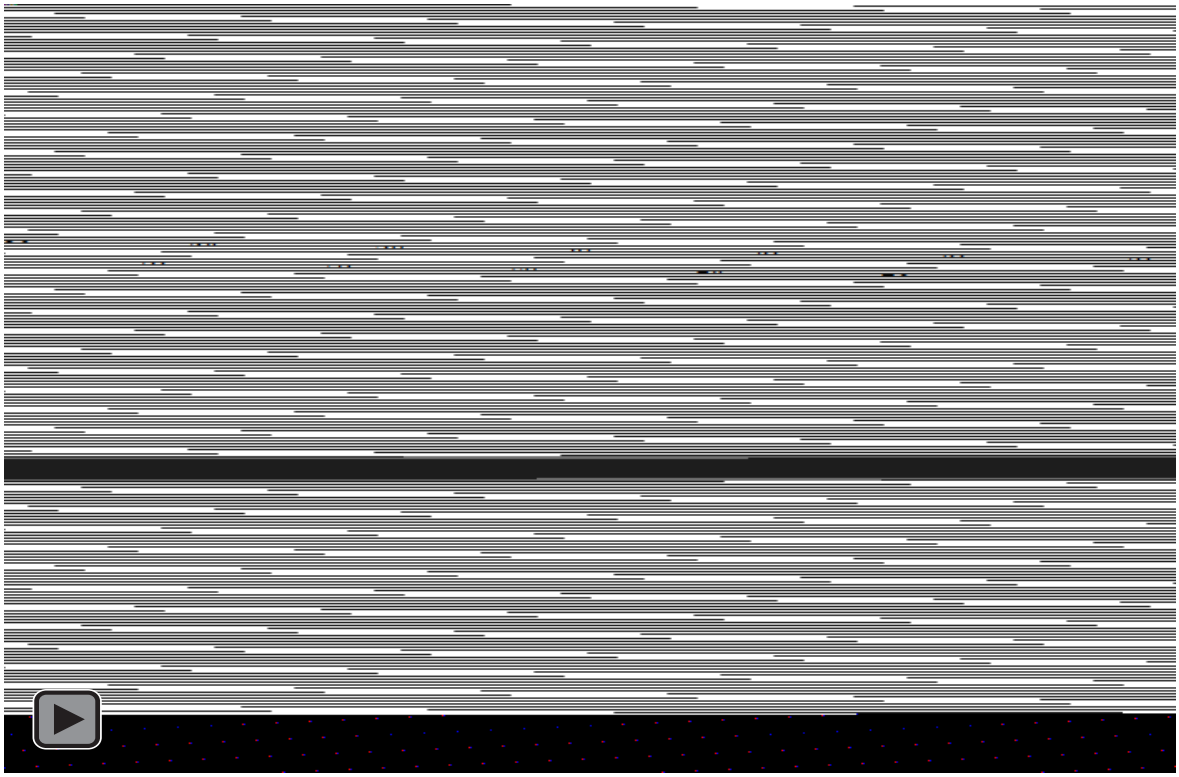
If you have to download a folder, you will need to use the `-r` argument like that:

```
sftp> get -r remote_dirpath local_dirpath
```

On the contrary, if you have to upload data from your local machine to the cluster, you will need to use the `put` command:

```
sftp> put local_filename_path remote_dirpath
```

In this video, you will see how to transfer a Python code and a bash file to psychp01 using `put`.



scp

`scp` (secure copy) is a command-line utility that allows you to securely copy files and directories between two locations. `scp` use requires a password, and both the files and password are encrypted so as to securely transfer data from one location to the other. `scp` uses the `ssh` protocol for both authentication and encryption. See here for my information.

When transferring data, `scp` takes on two main arguments:

```
scp source destination
```

The first argument is the address of the source file to transfer, the second the address where it has to be transferred to. A good way to memorize it is to think that `scp` needs to know `what` to send `where` to.

For example, to transfer files from the remote cluster (source) to your local machine (destination), use:

```
scp username@address_name:pathname_remote_src pathname_local_dest
```

Suppose my username (the one given to you by the IT when you got access to the cluster) is `gbellucci`, the filename of the file (e.g., a MATLAB file .m) I need to transfer is `best_analysis.m`, the pathname to that file on my local computer is `/Users/Gab`, and the pathname of the remote folder on the cluster I need to send my file to is `/home/gbellucci/coolest_project`. The line on terminal I need to transfer my file will be
To transfer files from your local machine to the remote cluster, use:

```
scp <space> pathname_local_src <space> username@address_name:pathname_remote_dest
```

Suppose the filename of the file (e.g., a MATLAB file .m) I need to transfer is `best_analysis.m`, the pathname to that file on my local computer is again `/Users/Gab`, and the pathname of the remote folder on the cluster I need to send my file to is `/home/gbellucci/coolest_project`. The command on the command line will be:

```
scp /Users/Gab/best_analysis.m gbellucci@psychp01.rhul.ac.uk:/home/gbellucci/coolest_project
```

Remember, your data will not be in your folder in the home directory but in your `MRIWork#` folder in `/MRIWork`. Hence, to upload a data file (say, `data.mat`), you'd need to type:

```
scp /Users/Gab/data.mat gbellucci@psychp01.rhul.ac.uk:/MRIWork/MRIWork25/data_coolest_project
```

If you have to upload or download multiple files or a file that contains multiple file (e.g., a folder), now you'll have a directory path (and not a file path), and you can use the `-r` argument to reiterate the sending over all files like that:

```
scp -r dirpath_local_src username@psychp01.rhul.ac.uk:dirpath_remote_dest
```

For example, if your directory path is to the folder called `analyses_folder`, you can type the following:

```
scp -r /Users/Gab/analyses_folder gbellucci@psychp01.rhul.ac.uk:/home/gbellucci/coolest_project
```

If you have a whole `data` folder to transfer, you will upload it into your `/MRIWork/MRIWork#` folder like that:

```
scp -r /Users/Gab/data gbellucci@psychp01.rhul.ac.uk:/MRIWork/MRIWork25/data_coolest_project
```

You would swap the two arguments if the folder is on the cluster, and you would need to get it onto your local computer:

```
scp -r username@psychp01.rhul.ac.uk:dirpath_remote_src dirpath_local_dest
```

For example, if your directory path is to the folder on the cluster called `results_folder` that you need to download into your `analyses_folder` on your local computer, you can type the following:

```
scp -r gbellucci@psychp01.rhul.ac.uk:/home/gbellucci/results_folder /Users/Gab/analyses_folder
```

rsync

`rsync`, which stands for *remote sync*, is a remote and local file synchronization tool. It uses an algorithm to minimize the amount of data copied by only moving the portions of files that have changed. Please see here for more information.

sshfs

`sshfs` allows you to mount the file system on your local machine. See here for more details. Basic usage for Linux users:

```
sshfs username@psychp01.rhul.ac.uk:dirpath mountpoint [options]
```

FileZilla

FileZilla is a free and open-source *File Transfer Protocol (FTP)* client that supports `ftp`, `ftps` and `sftp` protocols. It allows the implementation of the above command-line programs through a graphical interface. Please have a look at this step-by-step guide on how to use FileZilla.

ExpanDrive

An alternative to File Transfer Clients like the one mentioned above is ExpanDrive. ExpanDrive is a network filesystem client for MacOS, Microsoft Windows and Linux that facilitates mapping of local volume to many different types of cloud storage. It is different from the above File Transfer Clients because it is integrated into all applications on the operating system and does not require a file to be downloaded onto the local machine. On the contrary, remote files can be accessed, managed and changed as if they were stored locally.

The downside is that it is a non-free commercial tool.

Bash files and Batch system

Dos and don'ts

- Never run calculations on the home disk
- Always use the SLURM queueing system
- The login via the GUI is only for editing files, submitting jobs and visual inspections
- Do not run calculations interactively via the GUI
- Never use spaces for file names, folder names or job process names.

Batch System

The Psychp01 cluster is a resource that is shared between many of users and to ensure fair use everyone must do their computations by submitting jobs through a batch system that will execute the applications on the available resources.

The batch system on Psychp01 is SLURM (Simple Linux Utility for Resource Management).

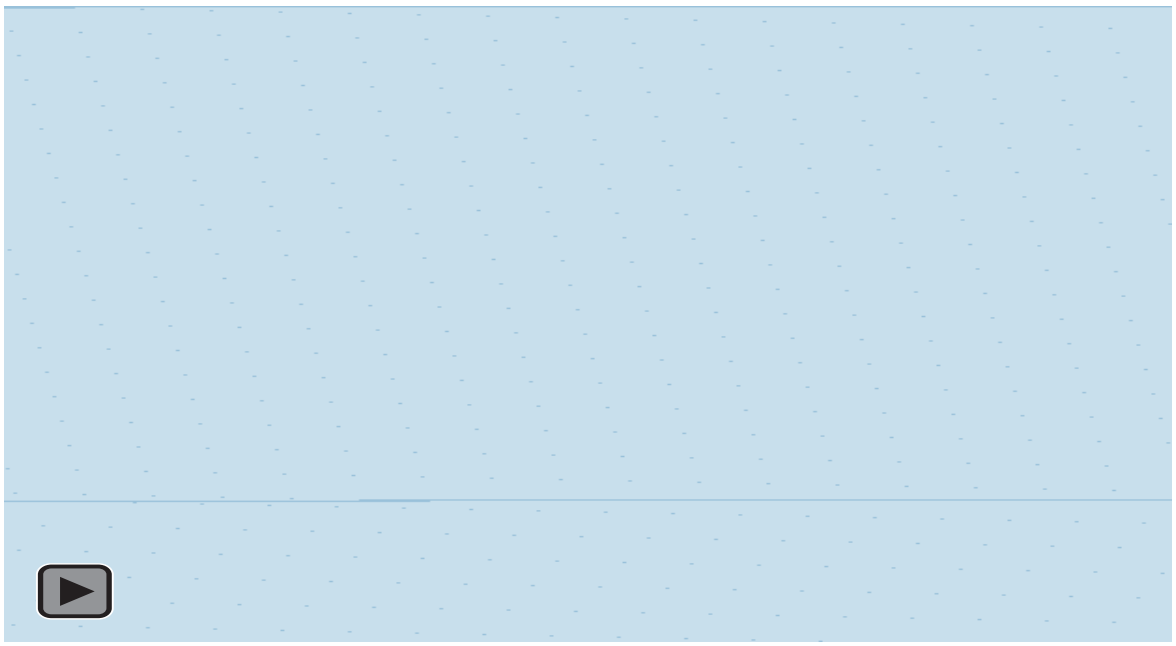
Creating a job script

To run a job on the batch system you need to create a job script. A job script is a regular shell script (bash) with some directives specifying the number of CPUs, memory, etc., that will be interpreted by the batch system upon submission.

After you wrote your job script as shown in the examples, you can start it with:

```
sbatch my_job_script_name.sh
```

You can find job script examples in job script examples. Below, there is a short video on how to run a Python code using a job script. See here for a step-by-step explanation.



How to pass command-line parameters to the job script

It is sometimes convenient if you do not have to edit the job script every time you want to change the input file. Or perhaps you want to submit hundreds of jobs and loop over a range of input files. For this it is handy to pass command-line parameters to the job script. For how to use different parameters and to find a link to a list of possible parameters, see SLURM Parameters.

In SLURM you can do this:

```
 $ sbatch myscript.sh myinput myoutput
```

And then you can pick the parameters up inside the job script:

```
#!/bin/bash

#SBATCH ...
#SBATCH ...
...

# argument 1 is myinput
# argument 2 is myoutput
mybinary.x < ${1} > ${2}
```

For recommended sets of parameters see also Advanced SLURM and jobs examples.

Software on psychp01

Psychp01 is a Linux based platform and has neuroimaging software installed on it. The software suite installed on the cluster is continuously evolving according to user needs and requirements. If an user needs a particular software for a study, the user is encouraged to get in touch with the IT team or Gabriele Bellucci at the earliest convenience. They will run a feasibility analysis and install the required software as soon as possible. Following is a list of available software on the cluster at the moment:

- **MATLAB R2023a** (*/usr/local/apps/MATLAB*)
- **GNU Octave 6.2.0** (*/usr/share/octave*)
- **Python 2.7** (*/usr/bin/python2*)
- **Python 3.11** (*/usr/bin/python3*)
- **SPM12** (*/usr/local/apps/SPM*)
- **FSl 6.0.7** (*/usr/local/apps/fsl_6.0.7*)
- **R 4.0.4** (*/usr/bin/R*)

Advanced SLURM

This chapter sources from this SLURM page.

Psychp01 uses the SLURM scheduler for running jobs. The home page for SLURM is http://slurm.schedmd.com/, and it is used in many computing systems, such as MASSIVE and VLSCI. SLURM is an open-source workload manager designed for Linux clusters of all sizes. It provides three key functions.

1. It allocates exclusive and/or non-exclusive access to resources (computer nodes) to users for some duration of time so they can perform work.
2. It provides a framework for starting, executing, and monitoring work (typically a parallel job) on a set of allocated nodes.
3. It arbitrates contention for resources by managing a queue of pending work.

The following material will explain how users can use SLURM. At the bottom of the page there is a PBS, SGE comparison section. SLURM Glossary It is important to understand that some SLURM syntax have meanings which may differ from syntax in other batch or resource schedulers.

SLURM: Glossary

Below is a summary of some SLURM concepts

Term	**Description**
Task	A task under SLURM is a synonym for a process, and is often the number of processes that are required
Success	A job completes and terminates well (with exit status 0) (cancelled jobs are not considered successful)
Socket	A socket contains one processor
Resource	A mix of CPUs, memory and time
Processor	A processor contains one or more cores
Partition	SLURM groups nodes into sets called partitions. Jobs are submitted to a partition to run. In other batch systems the term queue is used
Node	A node contains one or more sockets
Failure	Anything that lacks success
CPU	The term CPU is used to describe the smallest physical consumable, and for multi-core machines this will be the core. For multi-core machines where hyper-threading is enabled this will be a hardware thread.
Core	A CPU core
Batch job	A chain of commands in a script file
Account	The term account is used to describe the entity to which used resources are charged to. This field is not used on the Monarch cluster at the moment.

SLURM: Useful Commands

What	SLURM command	Comment
Job Submission	sbatch jobScript	SLURM directives in the jobs script can also be set by command line options for sbatch.
Check queue	squeue or aliases sq SQ	You can also examine individual jobs, i.e. squeue -j 792412
Check cluster status	show_cluster	This is a nicely printed description of the current state of the machines in our cluster, built on top of the sinfo command.
Deleting an existing job	scancel jobID	
Show job information	scontrol show job jobID	Also try show_job for nicely formatted output.
Suspend a job	scontrol suspend jobID	
Resume a job	scontrol resume jobID	
Deleting parts of a job array	scancel jobID_[5-10]	

SLURM: More on Shell Commands

Users submit jobs to psychp01 using SLURM commands called from the Unix shell (such as bash, or csh). Typically, a user creates a batch submission script that specifies what computing resources they want from the cluster, as well as the commands to execute when the job is running. They then use sbatch filename to submit the job. Users can kill, pause and interrogate the jobs they are running. Here is a list of common commands:

- To submit a job script for execution, use sbatch. The script will typically contain one or more srun commands to launch (parallel) tasks.

```
sbatch [options] job.sh
```

- To delete a job from the queue or stop it running, use `scancel`.

```
scancel jobID1 jobID2
scancel --name=[job name]
scancel                                          --user=[user]
```

- To report the state of partitions and nodes managed by SLURM, use `sinfo`. It has a wide variety of filtering, sorting, and formatting options.

```
sinfo [options]
```

Example:

```
gbellucci@psychp01:~$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
test*        up   infinite      1    mix psychp01
```

- To report the state of jobs or job steps use `squeue`. It has a wide variety of filtering, sorting, and formatting options. By default, it reports the running jobs in priority order and then the pending jobs in priority order.
    - For example, to print information only about a specific job step 65552.1:

```
squeue --steps 65552.1
STEPID     NAME  PARTITION    USER    TIME USE      NODELIST(REASON)
65552.1    test2     test   gabriele       12:49      dev[1-4]
```

    - We also have set up aliases to squeue that prints more information for them.

Alias	Maps to
sq	squeue -u <userid>
SQ	squeue -o"%.18i %.8P %.6a %.15j %.8u %8Q %.8T %.10M %.4c %.4C %.12l %.12L %.6D %.16S %.16V %R

    - SQ prints more information on the jobs, for all users and can be used like:

```
SQ -u myUserName
```

    - Some squeue options of interest. See *man squeue* for more information.

squeue option	Meaning
–array	Job arrays are displayed one element per line
–jobs=JobList	Comma separated list of Job IDs to display
–long	Display output in long format
–name=NameList	Filter results based on job name
–partition=PartitionList	Comma separated list of partitions to display
–user=User	Display results based on the listed user name

- To report or modify details of a currently running job, use `scontrol`.

```
scontrol show job 71701  #report details of job whose jobID is 71701
scontrol show jobid -dd 71701 # report more details on this job
scontrol hold 71701 #hold a job, prevents it being scheduled for execution
sctonrol release 71701   #release a job that was previously held manually
```

- Use `sacct` to view details on finished jobs

```
sacct -l -j jobID
```

- The command `sstat` shows metrics from currently running jobs when given a job number. Note, you need to launch jobs with srun to get this information.

Help on shell commands

Users have several ways of getting information on shell commands.

- The commands have man pages (via the Unix manual), e.g., `man sbatch`
- The commands have built-in help options, e.g.,

```
sbatch --help
sbatch --usage
```

Most commands have options in two formats:

- single letter, e.g., **-N 1**
- verbose, e.g., **--nodes=1**

Note the double dash -- in the verbose format. A non-zero exit code indicates failure in a command.

Some default behaviours:

- SLURM processes launched with srun are not run under a shell, so none of the following are executed:~/.profile~/.baschrc~/.login
- SLURM exports user environment by default (or –export=NONE)
- SLURM runs in the current directory (no need to cd $WORKDIR)
- SLURM combines stdout and stderr and outputs directly (and naming is different). The SLURM stdout /stderr file will be appended, not overwritten (if it exists)
- SLURM is case insensitive (e.g. project names are lower case)

Batch Scripts

A job script has a header section which specifies the resources that are required to run the job as well as the commands that must be executed. An example script is shown below.

```
!/bin/env bash

#SBATCH --job-name=example
#SBATCH --time=01:00:00
#SBATCH --ntasks=10
#SBATCH --cpus-per-task=10
#SBATCH --mem=2000

module load intel
uname -a
srun uname -a
```

Here are some of the SLURM directives you can use in a batch script. `man sbatch` will give you more information.

SLURM directive	**Description**
–job-name=[job name]	The job name for the allocation, defaults to the script name.

–partition=[partition name]	Request an allocation on the specified partition. If not specified jobs will be submitted to the default partition.
–time=[time spec]	The total wall time for the job allocation.
–array=[job spec]	Submit a job array with the defined indices.
–dependency=[dependency list]	Specify a job dependency.
–nodes=[total nodes]	Specify the total number of nodes.
–ntasks=[total tasks]	Specify the total number of tasks.
–ntasks-per-node=[ntasks]	Specify the number of tasks per node.
–cpus-per-task=[ncpus]	Specify the number of CPUs per task.
–ntasks-per-core=[ntasks]	Specify the number of tasks per CPU core.
–export=,[variable|ALL|NONE]	Specify what environment variables to export. NOTE: SLURM will copy the entire environment from the shell where a job is submitted from. This may break existing batch scripts that require a different environment than say a login environment. To guard against this –export=NONE can be specified for each batch script.

Running jobs:
Examples

If you want to see a general blueprint to write your own bash script, jump here. For specific examples, go to R example, Python example, MATLAB example.

Below here, there are a few introductory sections for your general information on SLURM workload management system. For more details on SLURM visit the SLURM website or check out the Advanced SLURM documentation in this manual.

SLURM Workload Manager

SLURM is the workload manager and job scheduler.

There are two ways of starting jobs with SLURM; either interactively with `srun` or as a script with `sbatch`.

Interactive jobs are a good way to test your setup before you put it into a script or to work with interactive applications like MATLAB or Python. You immediately see the results and can check if all parts behave as you expected. See Interactive Jobs for more details.

Please note: at our site if you submit SLURM task arrays it is very important to throttle the number of tasks/CPUs dispatched or you will take all the available resources.

SLURM Parameters

SLURM supports a multitude of different parameters. This enables you to effectively tailor your script to your need when using Psychp01 but also means that is easy to get lost and waste your time and quota.

Note: everything that is preceded by # followed by a space will be taken as a comment. On the contrary, `#SBATCH` followed by dash or double dash will define the parameters for the job.

The following parameters can be used as command line parameters with sbatch and `srun` or in `jobscript`, see script examples below. To use it in a `jobscript`, start a newline with `#SBATCH` followed by the parameter. For example, if you want to give a name to your job script, use the argument `--job-name=`, for example:

```
#SBATCH --job-name=my_test_job_name
```

NOTE: Do not use spaces in the job name. Something like that: `#SBATCH --job-name=my test job name` would not work!

See here for list of commands and here for examples on how to use them in batch scripts.

Differences between CPUs and tasks

As a new user writing your first SLURM job script the difference between `--ntasks` and `--cpus-per-task` is typically quite confusing. Assuming you want to run your program on a single node with 16 cores which SLURM parameters should you specify?

The answer is it depends whether your application supports MPI. MPI (message passing protocol) is a communication interface used for developing parallel computing programs on distributed memory systems. This is necessary for applications running on multiple computers (nodes) to be able to share (intermediate) results.

To decide which set of parameters you should use, check if your application utilizes MPI and therefore would benefit from running on multiple nodes simultaneously. On the other hand, if you have an non-MPI enables application or made a mistake in your setup, it doesn't make sense to request more than one node.
Currently, psychp01 has only a single node, hence when running your analyses, you would need to carefully choose only how many CPUs your analysis (task) requires by using `--cpus-per-task`.

Job Blueprint

You can save the following example to a file (e.g., run.sh) on psychp01. Comment the two cp commands that are just for illustration purpose (lines 46 and 55) and change the SBATCH directives where applicable. You can then run the script by typing:

```
sbatch run.sh
```

Please note that all values that you define with `sbatch` directives are hard values. When you, for example, ask for 6000 MB of memory (--mem=6000MB) and your job uses more than that, the job will be automatically killed by the manager.

```
#!/bin/bash -l

##############################
#       Job blueprint        #
##############################

# Give your job a name, so you can recognize it in the queue overview
#SBATCH --job-name=example

# Define, how many nodes you need. For now, psychp01 has only 1 node with a
100 CPUs.
#SBATCH --nodes=1
# By default, SLURM will allocate 1 CPU to your tasks. You can define the
number of CPUs you would need for your job as follows:

#SBATCH --cpus-per-task=20

# You can further define the number of tasks with --ntasks-per-*
# See "man sbatch" for details. e.g., --ntasks=4 will ask for 4 cpus.
# Define, how long the job will run in real time. This is a hard cap
meaning
```

```
# that if the job runs longer than what is written here, it will be
# force-stopped by the server. If you make the expected time too long, it
will
# take longer for the job to start. Here, we say the job will take 5
minutes.
#              d-hh:mm:ss

#SBATCH --time=0-00:05:00

# Define the partition on which the job shall run. May be omitted.
#SBATCH --partition=test

# How much memory you need.
# --mem will define memory per node and
# --mem-per-cpu will define memory per CPU/core. Choose one of those.
#SBATCH --mem-per-cpu=1500MB
##SBATCH --mem=5GB    # this one is not in effect, due to the double hash

# Turn on mail notification. There are many possible self-explaining
values:
# NONE, BEGIN, END, FAIL, ALL (including all aforementioned)
# For more values, check "man sbatch"
#SBATCH --mail-type=END,FAIL

# You may not place any commands before the last SBATCH directive

# Define and create a unique scratch directory for this job
SCRATCH_DIRECTORY=/home/${USER}/${SLURM_JOBID}
mkdir -p ${SCRATCH_DIRECTORY}
cd ${SCRATCH_DIRECTORY}

# You can copy everything you need to the scratch directory
# ${SLURM_SUBMIT_DIR} points to the path where this script was submitted
from
cp ${SLURM_SUBMIT_DIR}/myfiles*.txt ${SCRATCH_DIRECTORY}

# This is where the actual work is done. In this case, the script only
waits.
# The time command is optional, but it may give you a hint on how long the
# command worked
time sleep 10
#sleep 10

# After the job is done we copy our output back to $SLURM_SUBMIT_DIR
cp ${SCRATCH_DIRECTORY}/my_output ${SLURM_SUBMIT_DIR}

# In addition to the copied files, you will also find a file called
# slurm-1234.out in the submit directory. This file will contain all output
that
# was produced during runtime, i.e. stdout and stderr.

# After everything is saved to the home directory, delete the work
directory to
# save space on /home
cd ${SLURM_SUBMIT_DIR}
rm -rf ${SCRATCH_DIRECTORY}

# Finish the script
exit 0
```

Running many sequential jobs in parallel using job arrays

In this example we wish to run many similar sequential jobs in parallel using job arrays. We take Python as an example, but this does not matter for the job arrays:

```
#!/usr/bin/env python

import time

print('start at ' + time.strftime('%H:%M:%S'))

print('sleep for 10 seconds ...')
time.sleep(10)

print('stop at ' + time.strftime('%H:%M:%S'))
```

Save this to a file called “test.py” and try it out:

```
$ python test.py

start at 15:23:48
sleep for 10 seconds ...
stop at 15:23:58
```

Good. Now we would like to run this script 16 times at the same time. For this we use the following script:

```
#!/bin/bash -l

######################
# job-array example #
######################

#SBATCH --job-name=example

# 16 jobs will run in this array at the same time
#SBATCH --array=1-16

# run for five minutes
#              d-hh:mm:ss
#SBATCH --time=0-00:05:00

# 500MB memory per core
# this is a hard limit
#SBATCH --mem-per-cpu=500MB

# you may not place bash commands before the last SBATCH directive

# define and create a unique scratch directory
SCRATCH_DIRECTORY=/ptmp/${USER}/job-array-example/${SLURM_JOBID}
mkdir -p ${SCRATCH_DIRECTORY}
cd ${SCRATCH_DIRECTORY}

cp ${SLURM_SUBMIT_DIR}/test.py ${SCRATCH_DIRECTORY}

# each job will see a different ${SLURM_ARRAY_TASK_ID}
echo "now processing task id:: " ${SLURM_ARRAY_TASK_ID}
python test.py > output_${SLURM_ARRAY_TASK_ID}.txt
```

```
# after the job is done we copy our output back to $SLURM_SUBMIT_DIR
cp output_${SLURM_ARRAY_TASK_ID}.txt ${SLURM_SUBMIT_DIR}

# we step out of the scratch directory and remove it
cd ${SLURM_SUBMIT_DIR}
rm -rf ${SCRATCH_DIRECTORY}

# happy end
exit 0
```

Submit the script and after a short while you should see 16 output files in your submit directory.

R example

Running R scripts on psychp01 is very easy. Save the following R script in your home directory (e.g., `/home/gbellucci`) as `r_script.R`:

```
# Create a 2000x2000 matrix with random values
A1 <- matrix(runif(2000*2000), nrow = 2000, ncol = 2000)

# Perform 1000 fft operations on it
start_time <- as.numeric(Sys.time())
for (i in 1:1000) {
    A1 <- fft(A1)
}
time1 <- as.numeric(Sys.time()) - start_time

# write result time to file:
cat(paste('CPU time:', round(time1*1000,2),'ms'),file='results.txt')
```

Save the following job script in your home directory as `run_r_script.sh`.

```
#!/bin/bash -l
# My job script to run R code

#SBATCH -o ./job_output.%A_%a
#SBATCH -e ./job_errors.%A_%a
#SBATCH -D ./
#SBATCH -J run_r_test
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=10
#SBATCH --mem=6000
#SBATCH --time=00:05:00

# Use the command Rscript and set the path to the R script to run it. Change
/home/gbellucci with the location of your script on psychp01.
Rscript /home/gbellucci/r_script.R
```

Now on the command line, change your current directory to the home directory where you saved your `run_r_script.sh` and use the `sbatch` command to run the R script:

```
sbatch run_r_script.sh
```

When the job is finished you can display the output:

```
vim results.txt
```

Python example

Below is an example of a Python script to be used on the cluster. Save the following Python script in your home directory (e.g., `/home/gbellucci`) as `python_script.py`:

```
import numpy as np
from time import time

# Create a 2000x2000 matrix with random values
A1 = np.random.rand(2000, 2000).astype('float32')

# Perform 1000 fft operations on it
start_time = time()
for i in range(1000):
    A1 = np.fft.fft(A1)
end_time = time()
time1 = end_time - start_time

# write result time to file:
with open('results.txt', 'w') as fh:
    fh.write(f'CPU time: {time1*1000:.2f} ms\n')
```

The job script below will run your Python script. Save it in your home directory as `run_python_script.sh`.

```
#!/bin/bash -l
# My job script to run Python code

#SBATCH -o ./job_output.%A_%a
#SBATCH -e ./job_errors.%A_%a
#SBATCH -D ./
#SBATCH -J run_test
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=10
#SBATCH --mem=6000
#SBATCH --time=00:05:00

# import below here any python packages that are required for your script

# use the python command and set the path to your script to run it. Change
/home/gbellucci with the location of your script on psychp01.
python /home/gbellucci/python_script.py
```

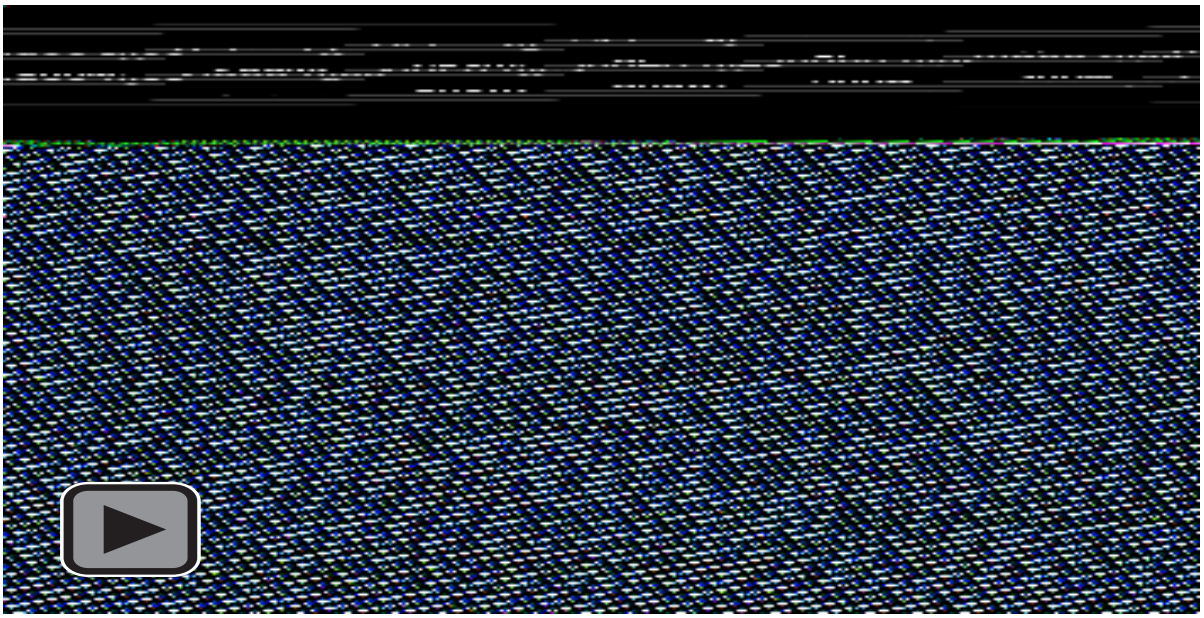
Now on the command line, change your current directory to the home directory where you saved your `run_python_script.sh` and use the `sbatch` command to run the Python script:

```
sbatch run_python_script.sh
```

When the job is finished you can display the output:

```
vim results.txt
```

Below, there is a short video that demonstrates what just described.



MATLAB example

Simple example

Running MATLAB scripts on psychp01 is pretty straightforward. Save the following MATLAB script in your home directory (e.g., `/home/gbellucci`) as `matlab_script.m`:

```
% Create a 2000x2000 matrix with random values
A1 = rand(2000,2000,'single');

% Perform 1000 fft operations on it
tic;
for i=1:1000
    A1 = fft(A1);
end
time1 = toc;

% write result time to file:
fh = fopen('results.txt','w+');
fprintf(fh,'CPU time: %.2f ms\n',time1*1000);
fclose(fh);
```

Save the following job script in your home directory as `run_matlab_script.sh`:

```
#!/bin/bash -l
# This is a comment for this job script to run the above matlab script

#SBATCH -o ./job_output.%A_%a
#SBATCH -e ./job_errors.%A_%a
#SBATCH -D ./
#SBATCH -J run_test_for_matlab_script
# --- resource specification (which resources for how long) ---
#SBATCH --partition=test
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=10
#SBATCH --mem=6000          # memory in MB required by the job
#SBATCH --time=00:05:00     # run time in h:m:s, up to 24h possible

# Use the srun command to run the MATLAB script. Change /home/gbellucci with
the location of your script on psychp01.
```

```
srun matlab -nodisplay -batch /home/gbellucci/matlab_script.m
```

Now on the command line, change your current directory to the home directory where you saved your `run_matlab_script.sh` and use the `sbatch` command to run the MATLAB script:

```
sbatch run_matlab_script.sh
```

When the job is finished you can display the output:

```
vim results.txt
```

Example with Parallel Computing

For **parallel computing**, we would need to modify our MATLAB script `matlab_script.m` to run processes in parallel as follows:

```
% Create a 2000x2000 matrix with random values
parpool(10);
A1 = rand(2000,2000,'single');

% Perform 1000 fft operations on it
tic;
parfor i=1:1000
    A1 = fft(A1);
end
time1 = toc;

% write result time to file:
fh = fopen('results.txt','w+');
fprintf(fh,'CPU time: %.2f ms\n',time1*1000);
fclose(fh);
```

No further changes are required to your job script `run_matlab_script.sh` in your home directory. You should only be sure that you are not requesting more CPUs in your MATLAB script than the ones you are allowing in the job script. That is, in MATLAB, command `parpool` should request either less than or the same number of CPUs as the ones you set in `#SBATCH --cpus-per-task=10`. Otherwise, your job will crash, as MATLAB will not find enough available CPUs when running.

Now run your job script as before using the `sbatch` command:

```
sbatch run_matlab_script.sh
```

Example with GPU Computing

For now, **GPU computing** is not supported on psychp01. However, in general, to run MATLAB processes on a GPU partition, we would need to add another block to our MATLAB script. Save it as `matlab_gpu.m`:

```
% Create a 2000x2000 matrix with random values
A1 = rand(2000,2000,'single');

% Perform 1000 fft operations on it
tic;
for i=1:1000
```

```
    A1 = fft(A1);
end
time1 = toc;

% copy the matrix onto the GPU
A2 = gpuArray(A1);
% perform the same 1000 operations
tic;
for i=1:1000
    A2 = fft(A2);
end
time2 = toc;

% write result time to file:
fh = fopen('results.txt','w+');
fprintf(fh,'CPU time: %.2f ms GPU time: %.2f ms Speedup factor:
%.2f\n',time1*1000,time2*1000,time1/time2);
fclose(fh);
```

We also have to modify the job script a bit. We will select the GPU partition. Save the new job script as `run_matlab_gpu_script.sh`:

```
#!/bin/bash -l
# This is a comment for this job script to run the above matlab script on
the GPU partition

#SBATCH -o ./job_output.%A_%a
#SBATCH -e ./job_errors.%A_%a
#SBATCH -D ./
#SBATCH -J mat_gpu
# --- resource specification (which resources for how long) ---
#SBATCH --partition=gpu
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=10
#SBATCH --mem=6000        # memory in MB required by the job
#SBATCH --time=00:05:00   # run time in h:m:s, up to 24h possible

# --- start from a clean state and load necessary environment modules ---

# run MATLAB
srun matlab -nodisplay -batch /home/gbellucci/matlab_gpu.m
```

And now run it like before:

```
sbatch run_matlab_gpu_script.sh
```

Interactive Jobs

Starting an interactive job

You can run an interactive job like this:

```
$ srun --nodes=1 --ntasks-per-node=1 --time=01:00:00 --pty bash -i
```

Here we ask for a single core on one interactive node for one hour with the default amount of memory. The command prompt will appear as soon as the job starts.
This is how it looks once the interactive job starts:

```
srun: job 12345 queued and waiting for resources
srun: job 12345 has been allocated resources
```

Exit the bash shell to end the job. If you exceed the time or memory limits the job will also abort.

Interactive jobs have the same policies as normal batch jobs, there are no extra restrictions. You should be aware that you might be sharing the node with other users, so play nice.

Keeping interactive jobs alive

Interactive jobs die when you disconnect from the login node either by choice or by internet connection problems. To keep a job alive, you can use a terminal multiplexer like `tmux`.

tmux allows you to run processes as usual in your standard bash shell. You start tmux on the login node before you get an interactive slurm session with `srun` and then do all the work in it.

In case of a disconnect you simply reconnect to the login node and attach to the tmux session again by typing:

```
tmux attach
```

or in case you have multiple session running:

```
tmux list-session
tmux attach -t SESSION_NUMBER
```

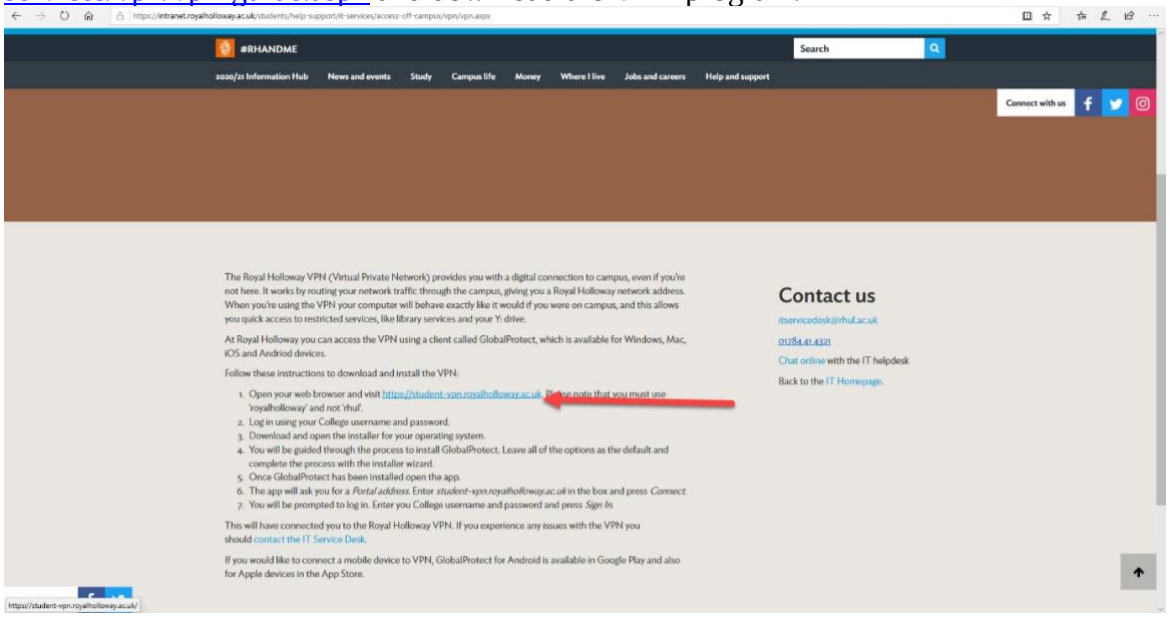
As long as the tmux session is not closed or terminated (e.g. by a server restart) your session should continue. One problem with tmux is that the tmux session is bound to the particular login server you get connected to. So, if you start a tmux session on a particular login node and next time you get randomly connected to a different login node, you first have to connect to back to the original login node. Given that psychp01 has currently only one node, this is not an issue for us now.

To log out a tmux session without closing it you have to press CTRL-B (that is, the Ctrl key and simultaneously “b”, which is the standard tmux prefix) and then “d” (without the quotation marks). To close a session just close the bash session with either CRTL-D or type exit.

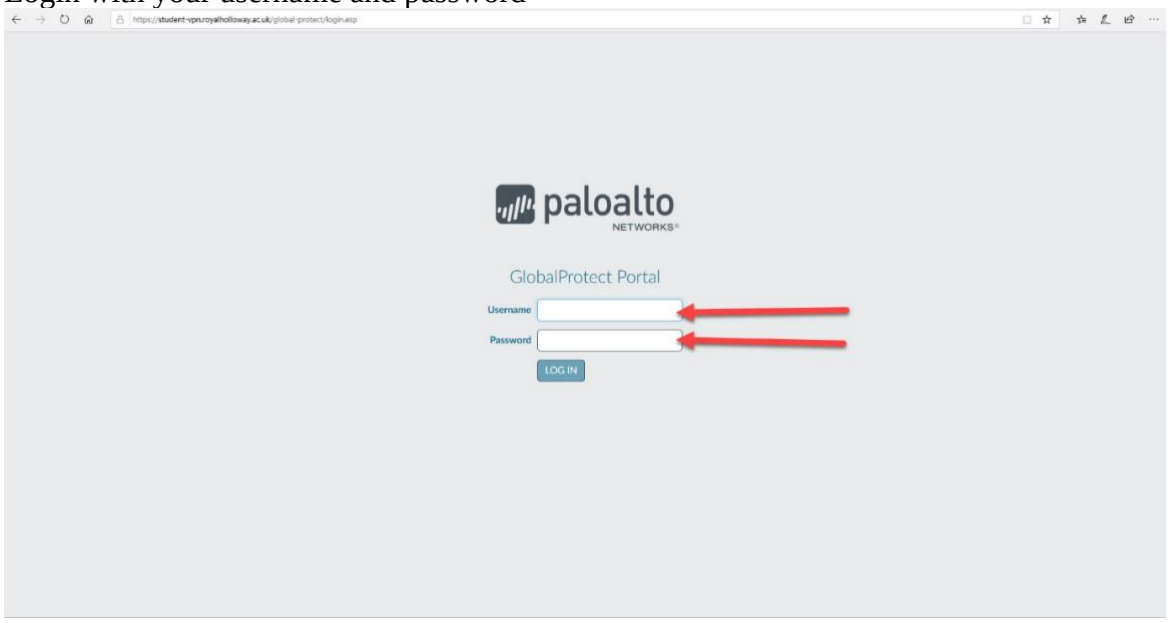
You can get a list of all tmux commands by CTRL-B and the ? (question mark). See also this page for a short tutorial of tmux. Otherwise working inside of a tmux session is almost the same as a normal bash session.

Installing VPN Client – GlobalProtect

Staff should go to the page https://intranet.royalholloway.ac.uk/staff/public/it-services/vpn/vpn-guide.aspx and download the VPN program.

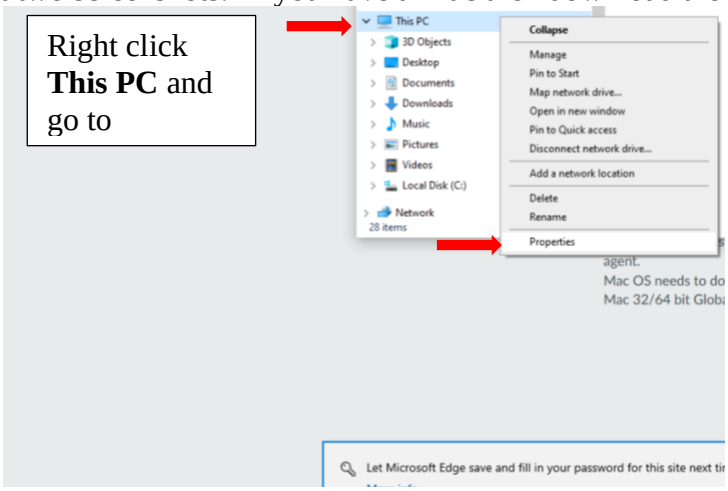


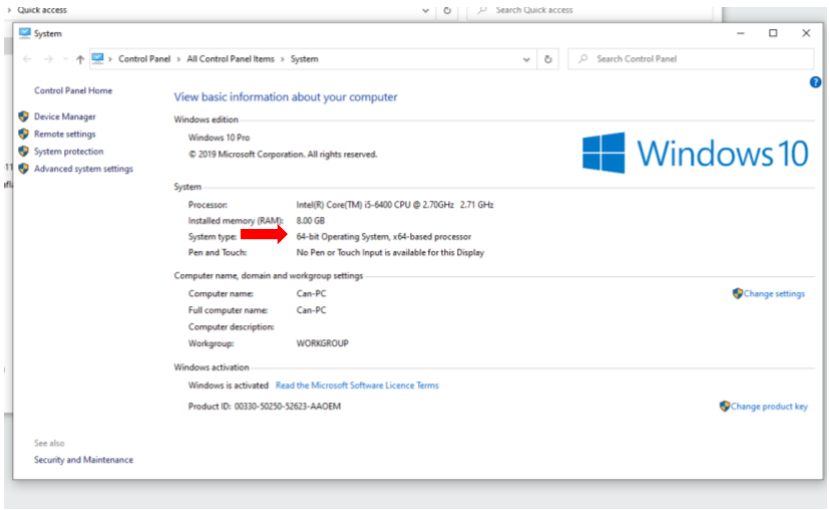
Login with your username and password

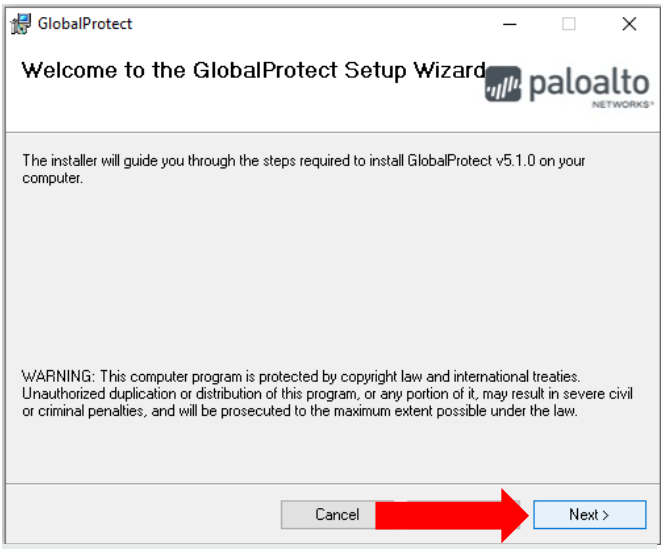


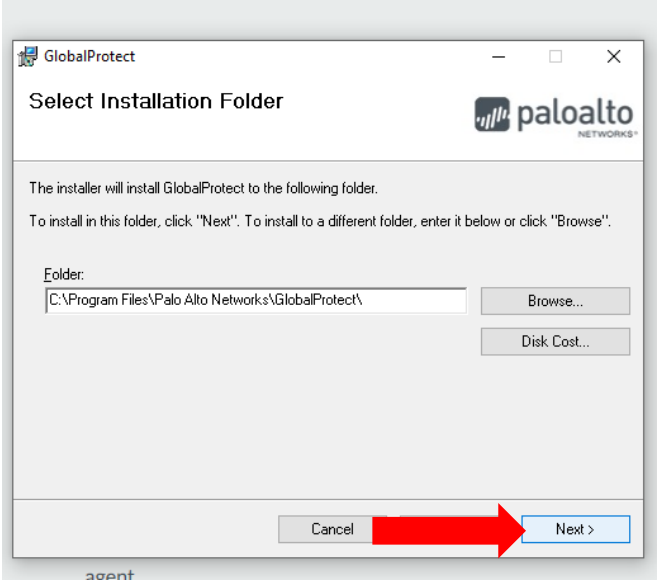
Download and install the GlobalProtect

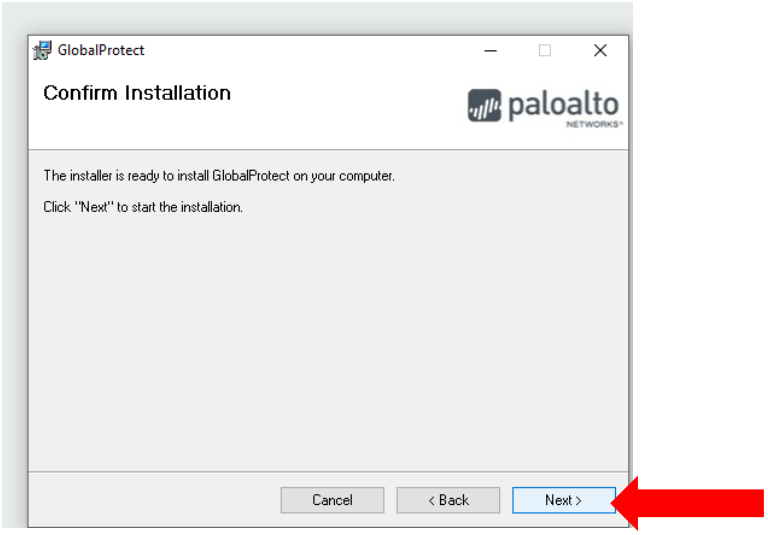
Most of you should have a 64 BIT version of Windows. To find out the version please follow the next two screenshots. If you have a Mac then download the Mac version.

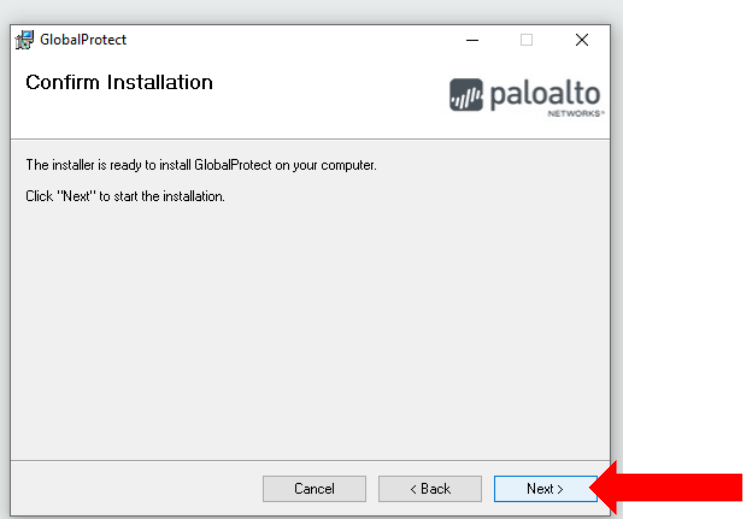




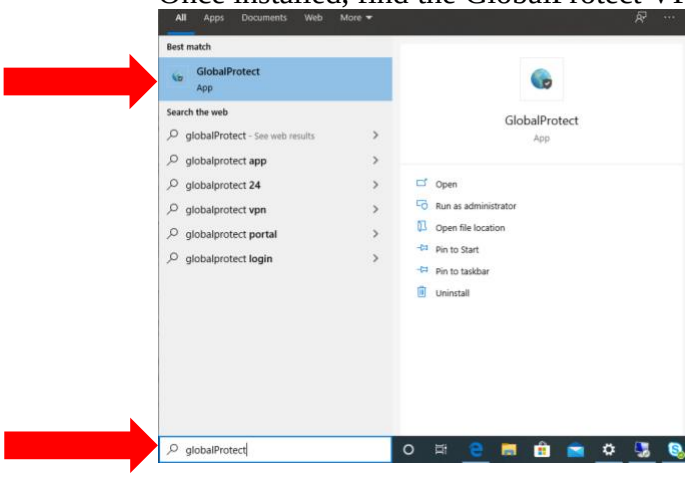

GlobalProtect
Welcome to the GlobalProtect Setup Wizard
paloalto
NETWORKS
The installer will guide you through the steps required to install GlobalProtect v5.1.0 on your computer.
WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.
Cancel
Next >


GlobalProtect
Select Installation Folder
paloalto
NETWORKS
The installer will install GlobalProtect to the following folder.
To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".
Folder:
C:\Program Files\Palo Alto Networks\GlobalProtect\
Browse...
Disk Cost...
Cancel
Next >

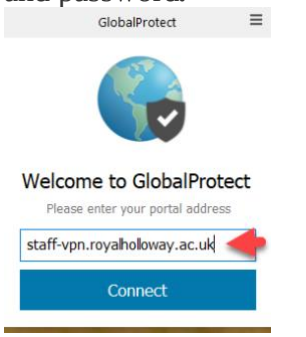

GlobalProtect
Confirm Installation
paloalto
NETWORKS
The installer is ready to install GlobalProtect on your computer.
Click "Next" to start the installation.
Cancel
< Back
Next >

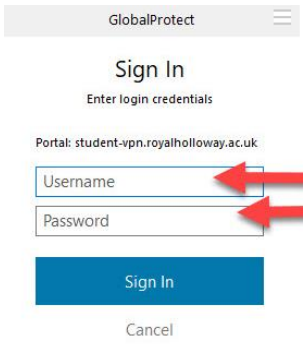


Once installed, find the GlobalProtect VPN program from the Start menu



Enter the server information ***staff-vpn.royalholloway.ac.uk*** and log in with your username and password**.**





Frequently Asked Questions

What are you looking for?

1. How do I access the cluster?
2. How do I transfer data and code to the cluster?
3. How do I run analyses on the cluster?
4. Which software is installed on the cluster?
5. How does SLURM work?