Server & HPC Guide at NHM

Erik Kusch (PhD)
www.erikkusch.com | erik.kusch@nhm.uio.no
Senior Engineer & Research Infrastructure Manager,
Biodiversity Digital Twin (BioDT),
Machine Readable Nature Research Group (MaNa),
Natural History Museum,
University of Oslo

Contents

Choosing the Right Service for You

Here at UiO, we have access to a host of computational resources. Choosing from these can be a bit a chore and will determine which sections of this guide will be useful to you. UiO itself has prepared a guide to this end: https://www.uio.no/english/services/it/research/hpc/find-service/.

Personally, I find that I use almost exclusively the windows statistic severs and the lightweight, free HPC resources at UiO for my work. To choose between these two, you can ponder these two questions:

1. Am I comfortable accessing computational resources through a terminal?
2. Do I require a large number of cores (>20) or RAM (>100 GB) for my computations?

If you answer "yes" to both of these, I strongly suggest you use the HPC resources at UiO. If any of your answer is a "no", I would instead suggest you use the windows statistic servers.

Disclaimer about Code Considerations

This computational resource guide is aimed at employees and students affiliated with the Natural History Museum at the University of Oslo. As such, I expect most readers will be predominantly interested in R coding. Therefore, all explorations of code execution herein will be focussed on R code. Nevertheless, the resources I highlight here can be used for more software than "just" the R environment. At the end of this document, you will find a small collection of useful commands for the different code environments this guide touches on.

UiO Windows Statistic-Server

The UiO windows statistic servers provide a familiar, powerful environment packaged up in a virtual desktop layout. You can read more about them here.

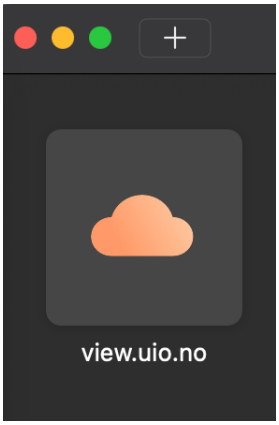
Registration & Getting Access

Anyone with a UiO username and password has access to the servers. You can log in from any machine.

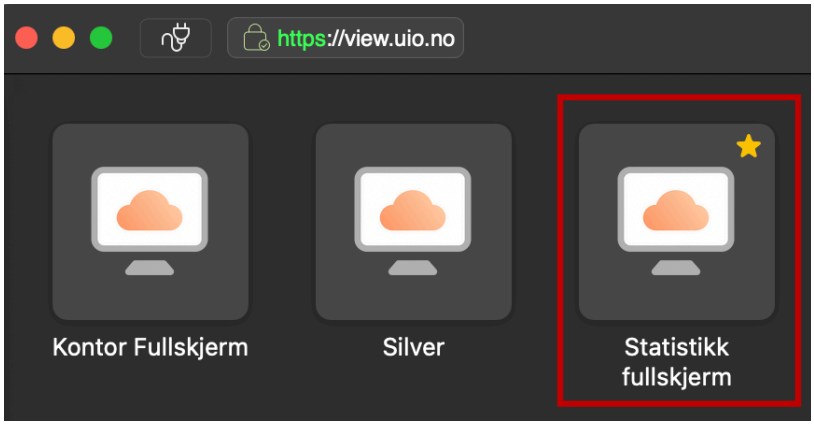
Connecting to the System

You can connect to the windows statistic servers either through your browser via this shortcut or, and preferably, through the VMWare Horizon software. This software should come pre-installed on any university-issued pc/mac. If it isn't preinstalled, follow this installation guide.

When opening VMWare Horizon, simply select the view.uio.no button. This will initialise the connection to the windows statistic servers. Once connected, you will see the below (as well as a number of additional programmes available to you – some of which we use for using the HPC resources at UiO). Connect to the statistic servers by clicking the "Statistikk Fullskjerm" button:



→



Once the connection to the statistic server is established, you will see the familiar Windows desktop:

Coding & Code Execution

R comes preinstalled on the windows statistic server. You can simply open it up and start coding. However, you need to manually connect it to GitHub, if you use that for your workflow.

When executing larger scripts that take considerable time to finish running, you can safely disconnect from the server – either through the windows-button in the taskbar (like you would normally power a windows PC off) or simply by closing the VMWare Horizon window. Your code will continue to execute even when you are disconnected.

However, if you start a job on the servers and disconnect, **you will remain logged in for 24 hours**. After that you will be logged out automatically, and any jobs you have running will be shut down. **If you wish to keep jobs running for longer, you need to log in again before 24 hours has passed**. If you are logged in but are not running any jobs, or using any resources, you will remain logged in for three days, after which you will be disconnected (but still logged in), and finally logged out after another 24 hours.

File Transfer

Lastly, to place data files and code scripts onto the windows statistic server or to harvest any data or code produced there, you will need to transfer files between the server and your local machine. How to do this the easiest way depends on your local operating system:

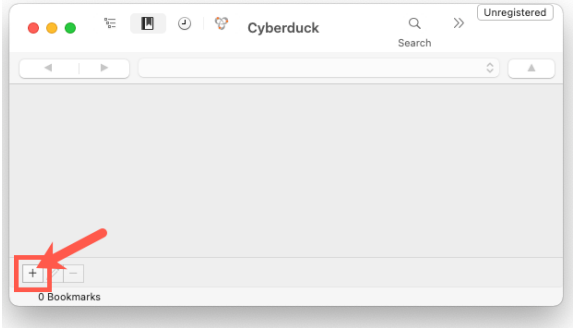
Windows

In windows machines, you can simply use the network drive that comes pre-registered to your device. This can be found in the "Your PC" view of the File Explorer. If this is missing, you can connect the desired network drive following these steps. You can now copy&paste and cut&paste freely between your local and the network drive. Alternatively, you may also follow the instructions for Mac below.

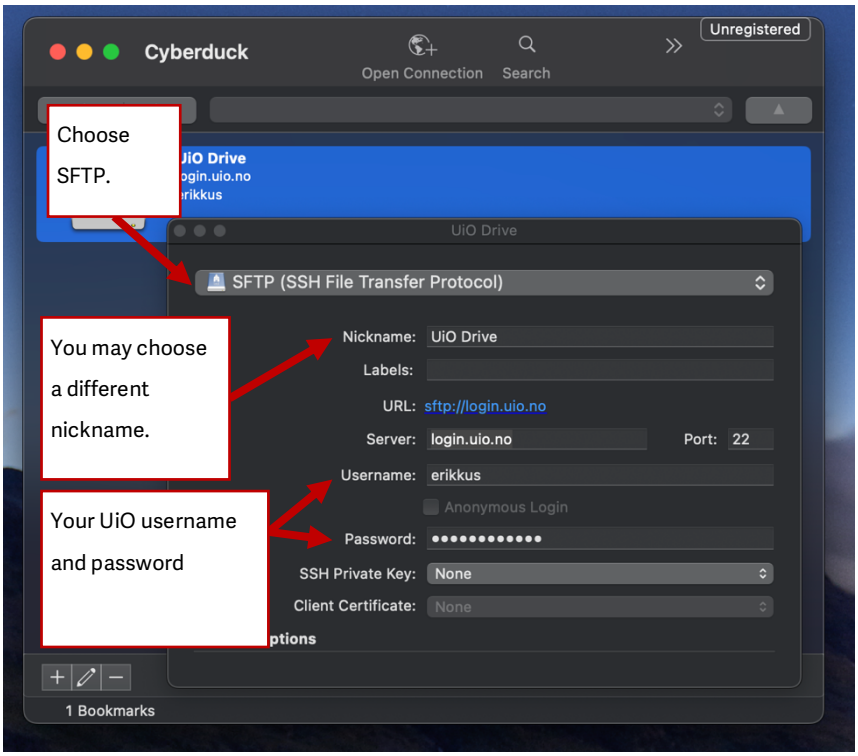
Mac

For file transfer on Mac, you are strongly recommended to use Cyberduck – a free-ware file-transfer program. It works also on windows. To set it up, you can follow this guide. Personally, I don't think this guide is the best though so I will show you how to set it all up right here.

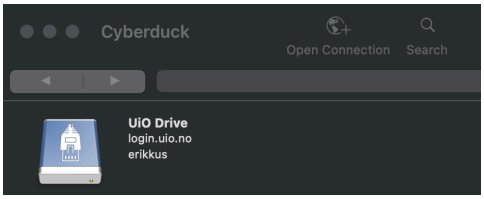
1. **Install Cyberduck.** On any UiO-issues machines, this should be pre-installed. On private machines, download and install from here. Please start Cyberduck now.
2. **Create a new connection.** You will need to register a new connection on Cyberduck. See the picture on the right-hand side for where to click:



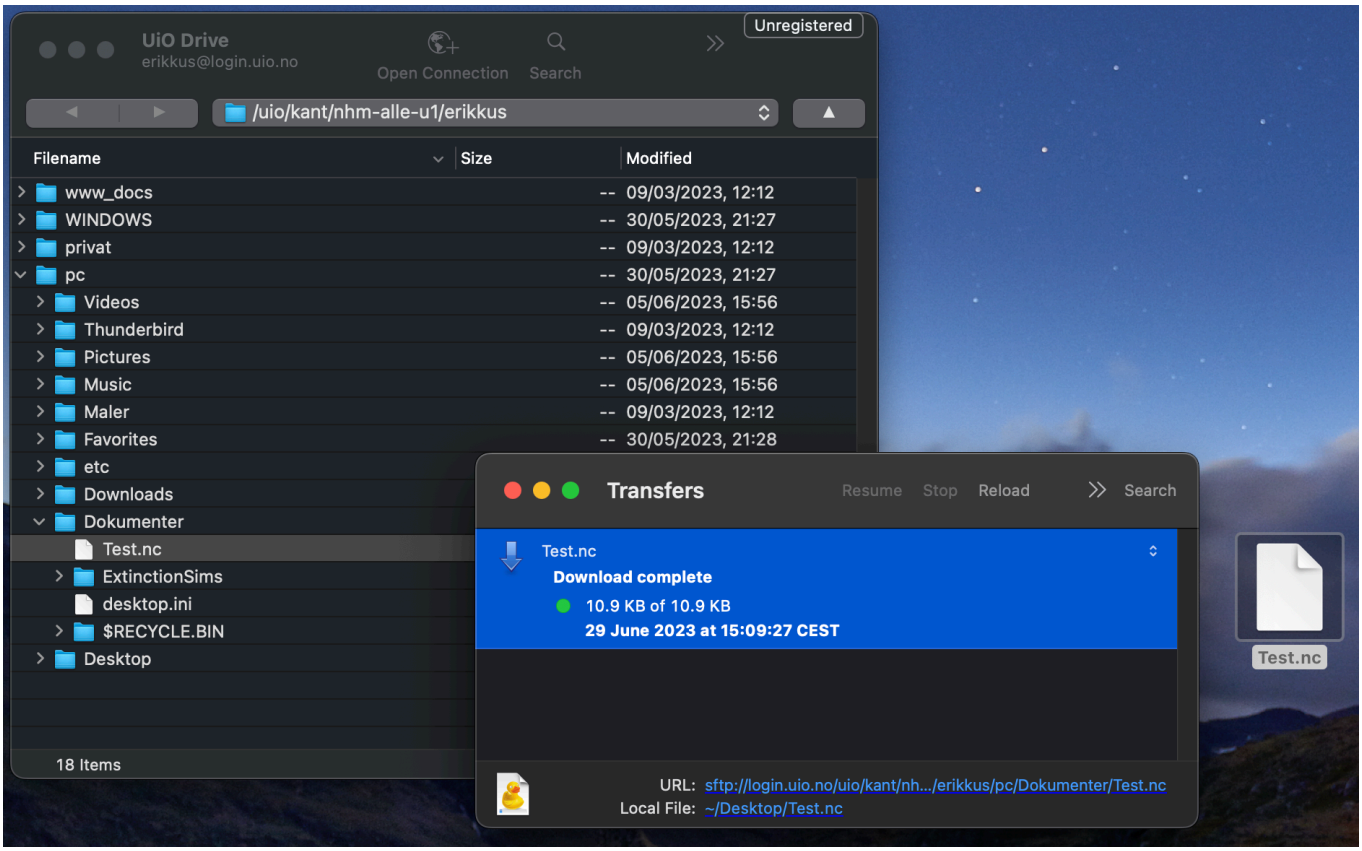
3. **Set-up the connection to the windows statistic server.** Next, you will need to set up the pointers and credentials for the connections to the windows statistic server drive. This is where my guide deviates from the UiO guide I linked above. Please fill in the fields as shown to the right:



4. **Connect to the windows statistic drive.** Simply click the connection icon in Cuberduck:



5. **Locate and transfer files.** Your files on the windows statistic server driver live under "pc" -> "Dokumenter". From there, you can drag and drop between device drives:



NHM HPC

At NHM, we have access to our very own High Performance Computing (HPC) service just for NHM affiliates. This is not a large HPC and shared between everyone at NHM. If you need larger capacities, please refer to the Free HPC services at UiO section.

Registration & Getting Access

To be granted access to this system, email hpc-drift@usit.uio.no and ask for an account to be created for you. If you are using an E-mail other than your ...@uio.no email, do inform them of your @uio.no email. Your username on the NHM HPC will be the same as your UiO identifier (the part before the @ in your uio.no email address).

Connecting to the System

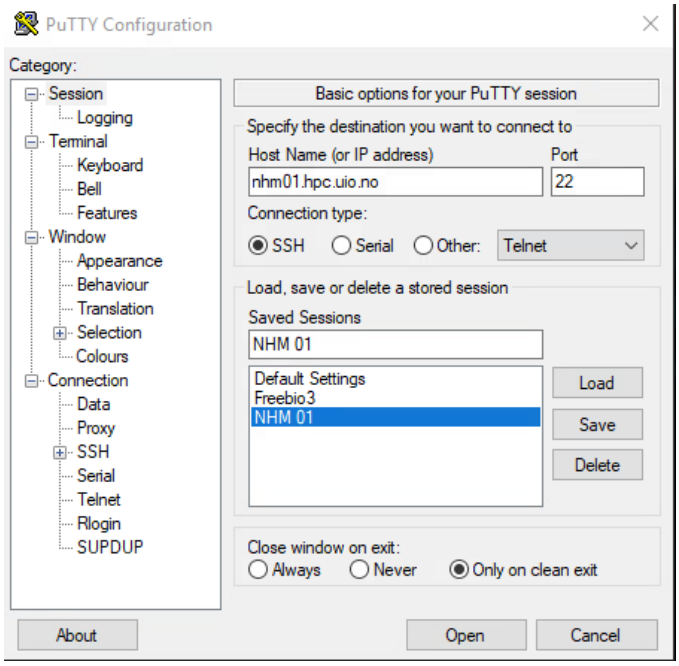
Once access has been granted and you are registered to use this resource, you must establish a connection with the server.. Since this has to be done through the university network, it is easiest to do so via the software provided directly through VMWare Horizon on UiO-issues devices or the webbrowser directly on all machines. Once there, we will start the PuTTY application (see right). Alternatively, you can also launch PuTTY from the windows statistic server.

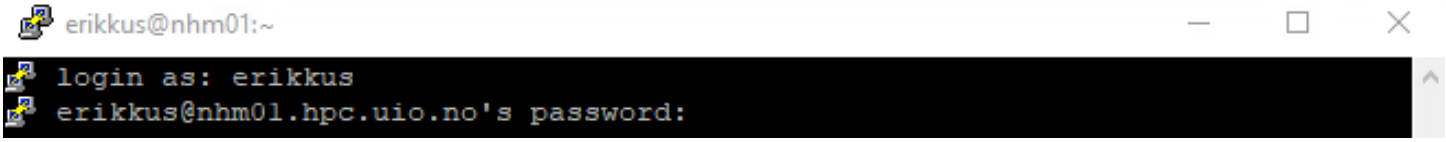
When PuTTY opens, register in the host name field nhm01.hpc.uio.no.

If you want to, you can now save this connection by clicking the save button and giving it a name.

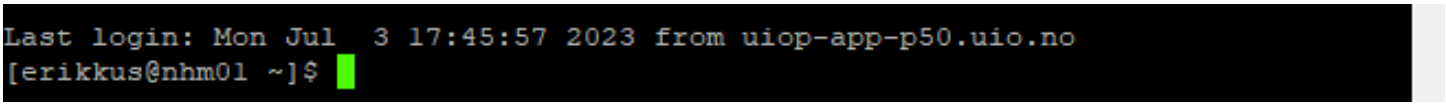
Hitting "Load" upon reopening PuTTY will automatically load the IP/Host Name.

Now click "Open" and enter yourt UiO username and password as prompted (you will not see it being typed) and hit RETURN when done with each:





You are now connected:



Coding & Code Execution

Code execution on HPC environments is a bit more involved than it is on servers like the windows statistic server at UiO.

Loading Software & Modules

To get started running code on such a server, we first need to start the code environment (R, in our case). To do so, we need to load the program itself into our personal HPC environment. This environment gets created as empty when we log on and we now want to load the R program into it. This is how new do it:

1. **Identify which version of the program is available to us.** This is done with the terminal command:
   module spider R
   Modules are what the HPC environment calls programs. Spider indicates for the HPC to look for something matching the writing following the word spider. The output we receive is:

```
[erikkus@nhm01 ~]$ module spider R

----------------------------------------------------------------------------
  R:
----------------------------------------------------------------------------
    Description:
      R is a free software environment for statistical computing and
      graphics.

     Versions:
        R/3.6.0-foss-2019a
        R/3.6.2-foss-2019b
        R/4.0.3-foss-2020a
        R/4.2.1-foss-2022a
     Other possible modules matches:
        AdapterRemoval  BRAKER  BayesTraits  Brotli  DendroPy  Evidencemodeler
...
```

   Swapping out R for something else, we can find other modules.

2. **Loading the program into our HPC Environment.** This is like loading libraries in R itself in that we make the module itself available to the HPC environment. We do this via the command: module load R/4.2.1-foss-2022a
   In the future, you may need to change the R version according to which versions you obtain with step 1 above:

```
[erikkus@nhm01 ~]$ module load R/4.2.1-foss-2022a
```

3. **Prepare library folder for R packages.** You need to prepare a local folder for R package installation. You can do so before opening the R environment by running the following two lines in the HPC terminal:
   export R_LIBS=~/local/rlibs
   mkdir -p ~/local/rlibs
4. **Start the module/R.** Now we can start the R environment via the terminal command: R
   Now, you find yourself in an R environment in which you can code like normal:

```
[erikkus@nhm01 ~]$ R

R version 4.2.1 (2022-06-23) -- "Funny-Looking Kid"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Interactive Coding

Following the procedure above, you find yourself in the familiar R console environment. Here you can code just like you would in an R console instance on your local machine. Some useful commands you should be aware of in this environment are:

`getwd()` – shows current working directory

`list.files()` – lists all files in current working directory

`source()` – executes a code file

`q()` – quits the R environment

Note that **any code run in this environment will terminate execution when you disconnect from the server**. To avoid this, you need to go through **unsupervised code execution**!

Unsupervised Code Execution

To ensure your code continues execution after you disconnect from the server, you need to create separate HPC environments within which to run your code and which you can safely detach without killing processes therein. There are several ways of doing this. I prefer **screen** environments for this task. To work with screen environments, follow these steps:

1. Open a screen via the terminal line: `screen`

```
[erikkus@nhm01 ~]$ screen
```

```
[screen 0: erikkus@nhm01:~]
[erikkus@nhm01 ~]$
```

2. Load modules and start R – we already covered this above.

```
[screen 0: erikkus@nhm01:~]
[erikkus@nhm01 ~]$ module load R/4.2.1-foss-2022a
[erikkus@nhm01 ~]$ R

R version 4.2.1 (2022-06-23) -- "Funny-Looking Kid"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

3. Execute code – this is completely up to you. As an example, for me it looks like this:

```
[Previously saved workspace restored]

> source("2 - Extinction Simulation.R")
```

4. Disconnect from the screen via the keyboard shortcut: Ctrl+a+d (press them all at the same time). This brings you back to the previous HPC environment:

```
erikkus@nhm01:~
[detached from 2422943.pts-2.nhm01]
[erikkus@nhm01 ~]$
```

Your code is still being executed in the screen environment we just disconnected from.

5. Re-open a screen. To re-open a screen to inspect code execution, for example, you must first identify which screens are open. This can be done via:

screen -ls or screen -r (if there are multiple screens in existence)

```
[erikkus@nhm01 ~]$ screen -ls
There is a screen on:
        2422943.pts-2.nhm01     (Detached)
1 Socket in /run/screen/S-erikkus.
[erikkus@nhm01 ~]$
```

Re-opening a specific screen is now as easy as executing:

screen -r XXXX (where XXX is the persistent unique identifier of each screen as shown above)

```
[erikkus@nhm01 ~]$ screen -r 2422943.pts-2.nhm01
```

This opens your screen environment back up with the R console in the foreground:

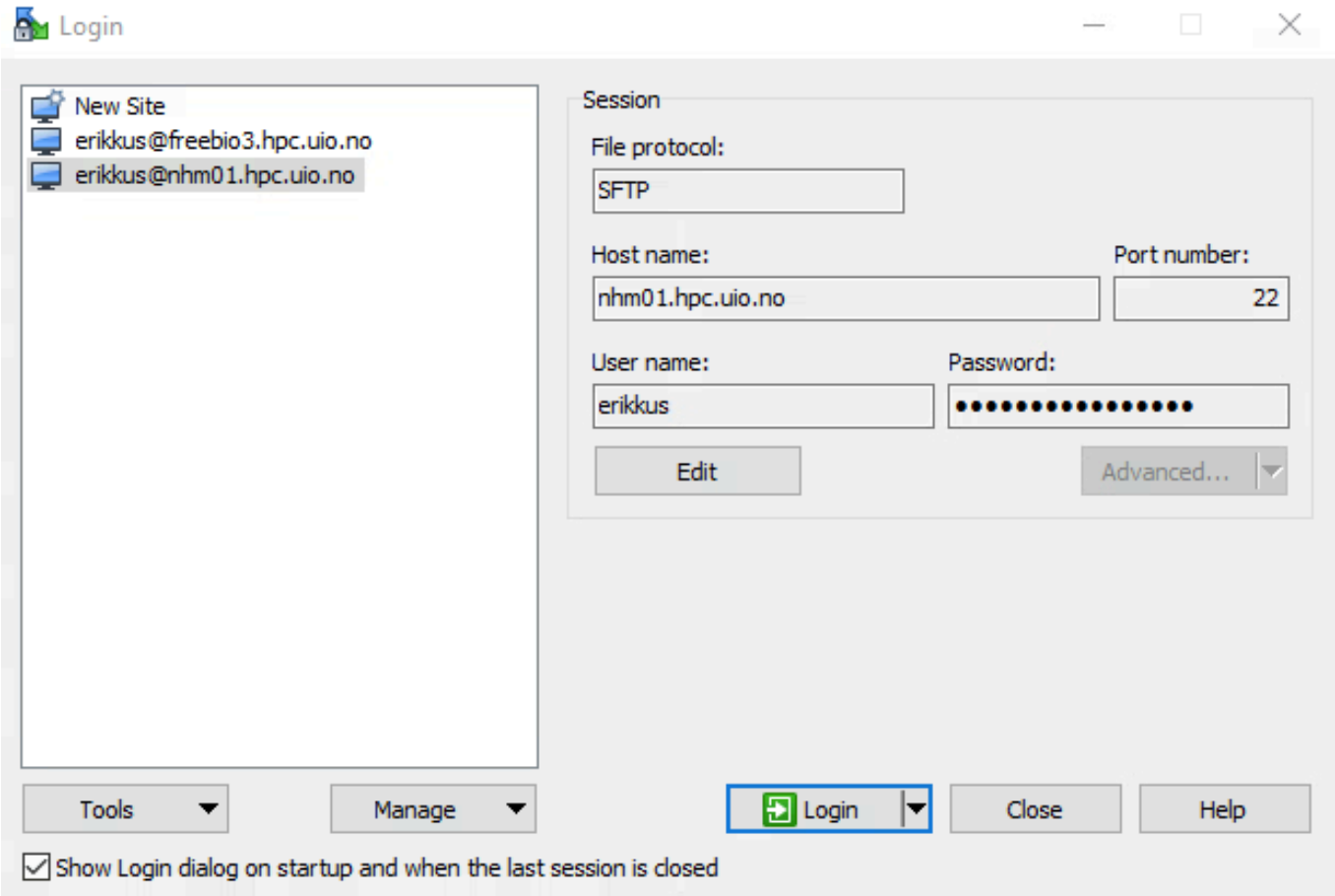
```
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=02m 44s
## Animals ##
IS (% of IS required for continued existence) = 0.35
Rewiring Cutoff (probability of rewiring required) = 0.8
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=01h 31m 16s
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=03m 36s
```

Note that the progress bars here are console outputs of one of my scripts.

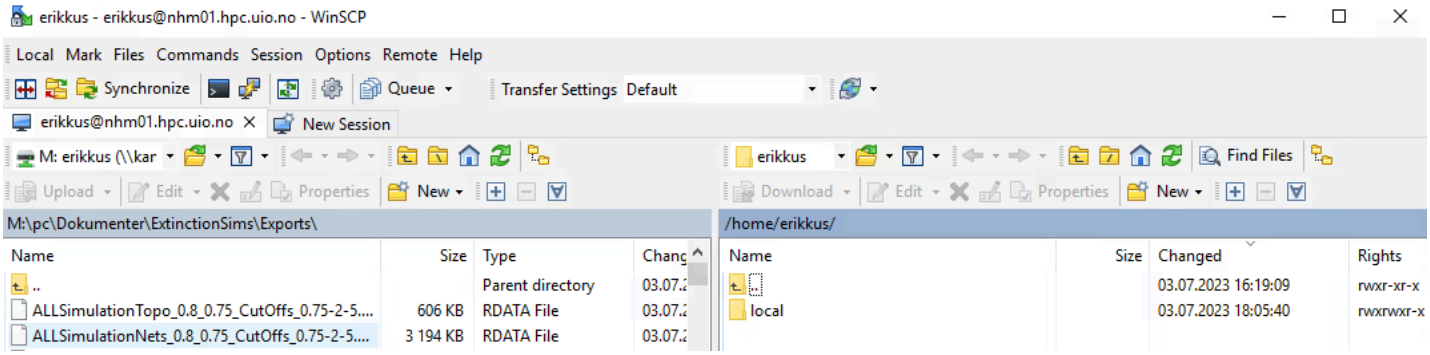
Using this screen procedure, you can disconnect from the server when not in a screen environment and code execution in that screen environment will continue. You can check that you are properly disconnected from each screen when the screen -ls command returns only (Detached) screens.

File Transfer

To inject files into the HPC system and harvest them from there, you can use WinSCP. There are more straightforward solutions available, but I have found this one to be the most user-friendly due to its reliance on a graphical user interface. Just like PuTTY (which we use for code execution in the HPC environment), WinSCP can be started via the VMWare Horizon client or the webbrowser. There, create a "Site" and enter the same host name as you did for your PuTTY session:



You may also enter your username and password if you want them saved for easier re-connections. After you click "Login", the connection is established and you are greeted with a two-pane window – the right-hand side is the server drive, the left-hand side is the windows statistic drive by default. You can now drag and drop between these drives. Alternatively, you can also drag and drop into any of these panes from your local machine. If you wish to change either drives and directories, you can do so via the respective dropdown boxes:

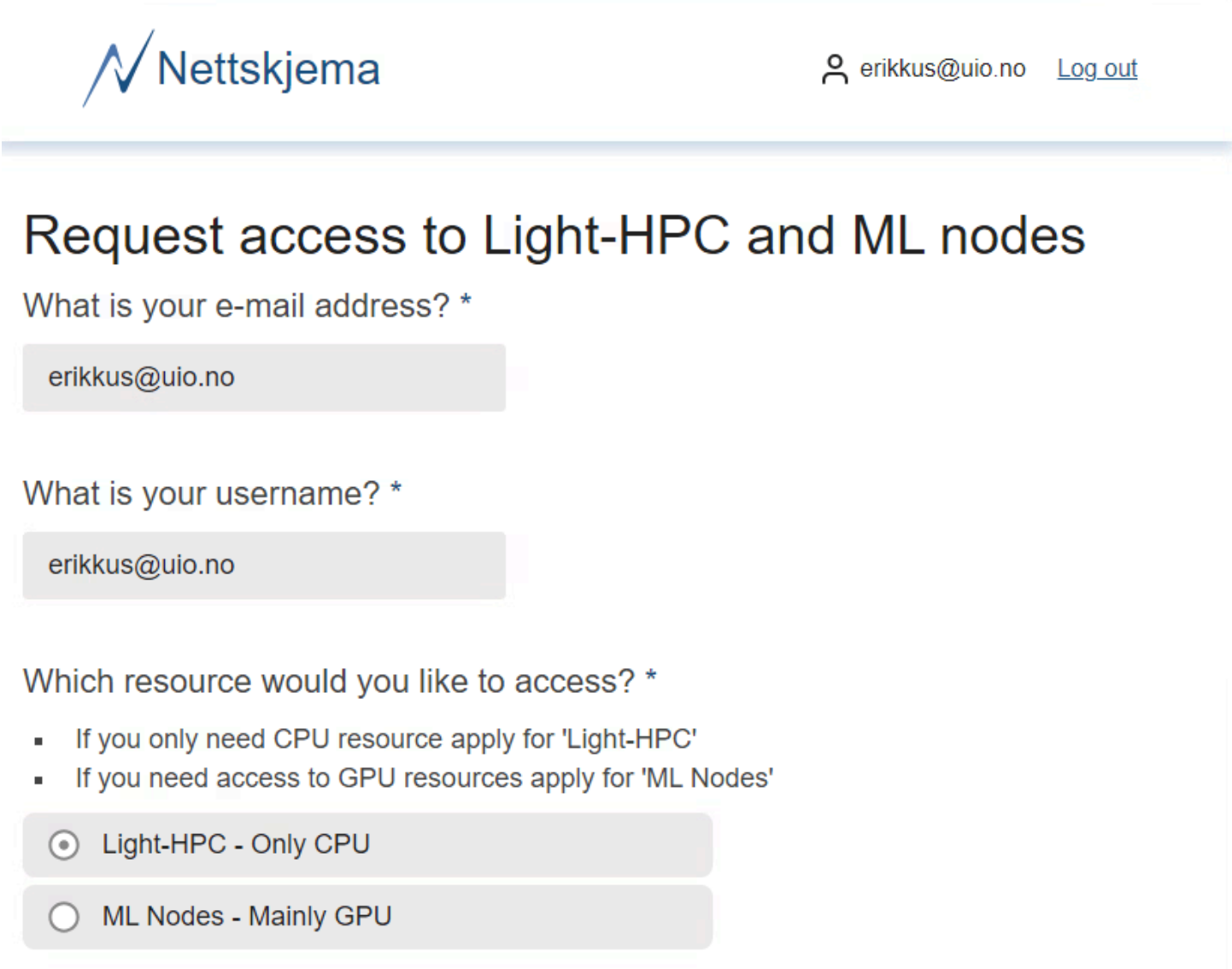


UiO Free HPC Services

There are several High Performance Computing (HPC) services available to UiO-affiliates. An overview of them is available here. Personally, I have found the Lightweight HPC resources to be plenty for my fairly intense computational demands. I prefer these over other HPC resources offered via UiO since they are free of charge and easy to access. **Therefore, this section of this guide deals exclusively with the Lightweight HPC resources at UiO.**

Registration & Getting Access

To be granted access to these resources, fill in this form. Turn-around times on access being granted are usually short. For my use, I asked for access to the CPU resources:



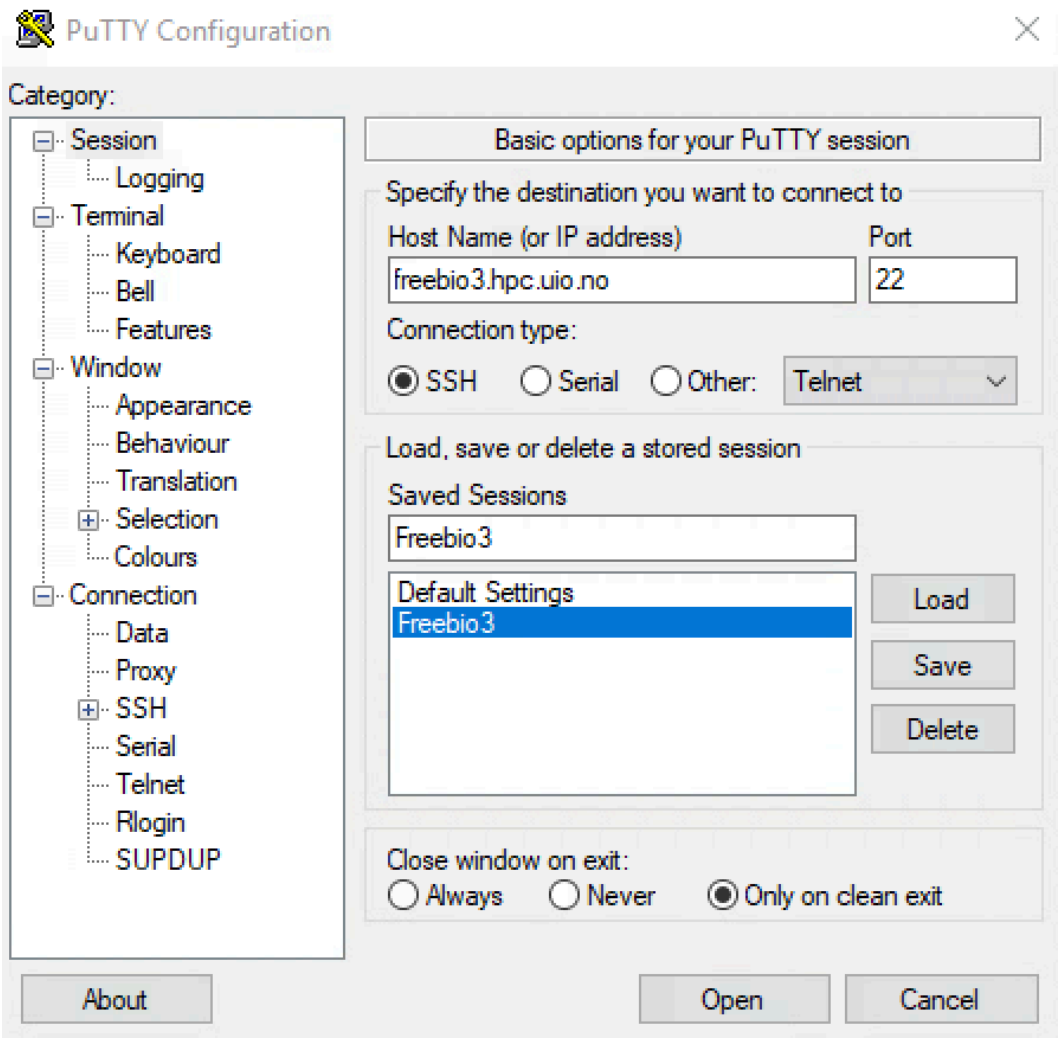
Now you simply wait to be granted access.

Connecting to the System

Once access has been granted and you are registered to use these resources, you must establish a connection with the server(s). Since this has to be done through the university network, it is easiest to do so via the software provided directly through VMWare Horizon on UiO-issues devices or the webbrowser directly on all machines. Once there, we will start the PuTTY application (see right). Alternatively, you can also launch PuTTY from the windows statistic server.

When PuTTY opens, register in the host name field one of the following specifications depending on which server you want to use (see their specifications here):

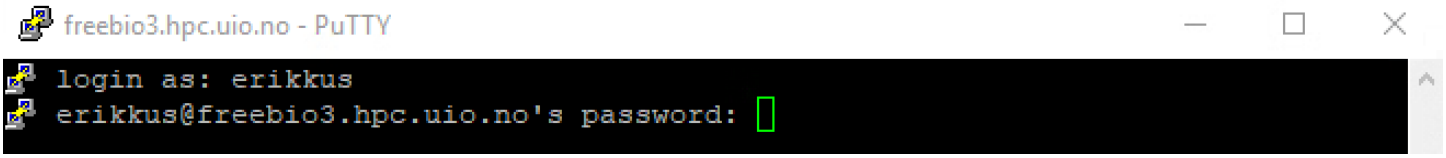
- freebio1.hpc.uio.no
- freebio2.hpc.uio.no
- freebio3.hpc.uio.no
- freebio4.hpc.uio.no
- biont01.hpc.uio.no
- biont02.hpc.uio.no
- biont03.hpc.uio.no
- biont04.hpc.uio.no



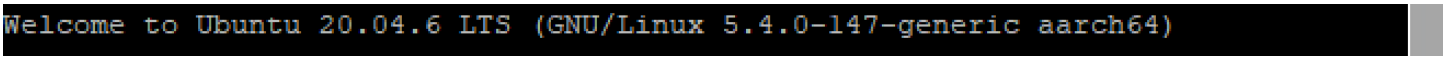
If you want to, you can now save this connection by clicking the save button and giving it a name.

Hitting "Load" upon reopening PuTTY will automatically load the IP/Host Name.

Now click "Open" and enter yourt UiO username and password as prompted (you will not see it being typed) and hit RETURN when done with each:



You are now connected:



Coding & Code Execution

Code execution on HPC environments is a bit more involved than it is on servers like the windows statistic server at UiO.

Loading Software & Modules

To get started running code on such a server, we first need to start the code environment (R, in our case). To do so, we need to load the program itself into our personal HPC environment. This environment gets created as empty when we log on and we now want to load the R program into it. This is how new do it:

5. **Identify which version of the program is available to us.** This is done with the terminal command:
   module spider R
   Modules are what the HPC environment calls programs. Spider indicates for the HPC to look for something matching the writing following the word spider. The output we receive is:

```
erikkus@freebio3:~$ module spider R

-----------------------------------------------------------------------------------------------------------------
  R:
-----------------------------------------------------------------------------------------------------------------
    Description:
      R is a free software environment for statistical computing and graphics.

     Versions:
        R/4.2.0-foss-2021b
        R/4.2.2-foss-2022b
     Other possible modules matches:
        Armadillo  Arrow  Brotli  Brunsli  FriBidi  GCCcore  GObject-Introspection  Ghostscript  HarfBuzz  JasPer  ...

-----------------------------------------------------------------------------------------------------------------
  To find other possible module matches execute:

      $ module -r spider '.*R.*'

-----------------------------------------------------------------------------------------------------------------
  For detailed information about a specific "R" package (including how to load the modules) use the module's full name.
  Note that names that have a trailing (E) are extensions provided by other modules.
  For example:

     $ module spider R/4.2.2-foss-2022b
-----------------------------------------------------------------------------------------------------------------
```

   Swapping out R for something else, we can find other modules.

6. **Loading the program into our HPC Environment.** This is like loading libraries in R itself in that we make the module itself available to the HPC environment. We do this via the command: module load R/4.2.2-foss-2022b
   In the future, you may need to change the R version according to which versions you obtain with step 1 above:

```
erikkus@freebio3:~$ module load R/4.2.2-foss-2022b
```

7. **Prepare library folder for R packages.** You need to prepare a local folder for R package installation. You can do so before opening the R environment by running the following two lines in the HPC terminal:
   export R_LIBS=~/local/rlibs
   mkdir -p ~/local/rlibs
8. **Start the module/R.** Now we can start the R environment via the terminal command: R
   Now, you find yourself in an R environment in which you can code like normal:

```
erikkus@freebio3:~$ R

R version 4.2.2 (2022-10-31) -- "Innocent and Trusting"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: aarch64-unknown-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

>
```

Interactive Coding

Following the procedure above, you find yourself in the familiar R console environment. Here you can code just like you would in an R console instance on your local machine. Some useful commands you should be aware of in this environment are:

getwd() – shows current working directory

list.files() – lists all files in current working directory

source() – executes a code file

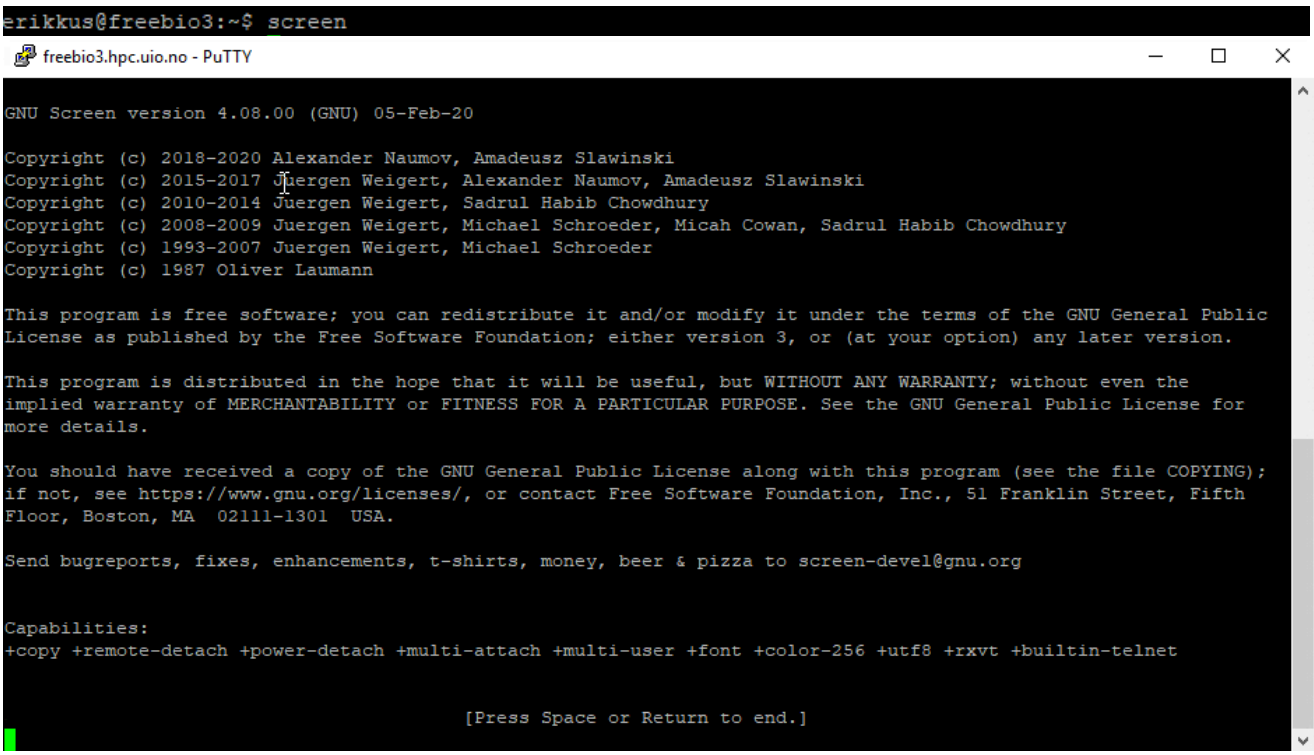
q() – quits the R environment

Note that **any code run in this environment will terminate execution when you disconnect from the server**. To avoid this, you need to go through **unsupervised code execution**!

Unsupervised Code Execution

To ensure your code continues execution after you disconnect from the server, you need to create separate HPC environments within which to run your code and which you can safely detach without killing processes therein. There are several ways of doing this. I prefer **screen** environments for this task. To work with screen environments, follow these steps:

6. Open a screen via the terminal line: screen



Now simply hit RETURN and you are in a screen environment:

7. Load modules and start R – we already covered this above.

```
erikkus@freebio3:~$ module load R/4.2.2-foss-2022b
erikkus@freebio3:~$ R

R version 4.2.2 (2022-10-31) -- "Innocent and Trusting"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: aarch64-unknown-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

>
```

8. Execute code – this is completely up to you. As an example, for me it looks like this:

```
[Previously saved workspace restored]

> source("2 - Extinction Simulation.R")
```

9. Disconnect from the screen via the keyboard shortcut: Ctrl+a+d (press them all at the same time). This brings you back to the previous HPC environment:

```
freebio3.hpc.uio.no - PuTTY
[detached from 3867178.pts-0.freebio3]
erikkus@freebio3:~$
```

Your code is still being executed in the screen environment we just disconnected from.

10. Re-open a screen. To re-open a screen to inspect code execution, for example, you must first identify which screens are open. This can be done via:

screen -ls or screen -r (if there are multiple screens in existence)

```
erikkus@freebio3:~$ screen -list
There are screens on:
        3922583.pts-3.freebio3   (06/21/2023 03:06:44 PM)        (Detached)
        3867178.pts-0.freebio3   (06/21/2023 05:59:44 AM)        (Detached)
2 Sockets in /run/screen/S-erikkus.
erikkus@freebio3:~$
```

Re-opening a specific screen is now as easy as executing:

screen -r XXXX (where XXX is the persistent unique identifier of each screen as shown above)

```
erikkus@freebio3:~$ screen -r 3922583.pts-3.freebio3
```

This opens your screen environment back up with the R console in the foreground:

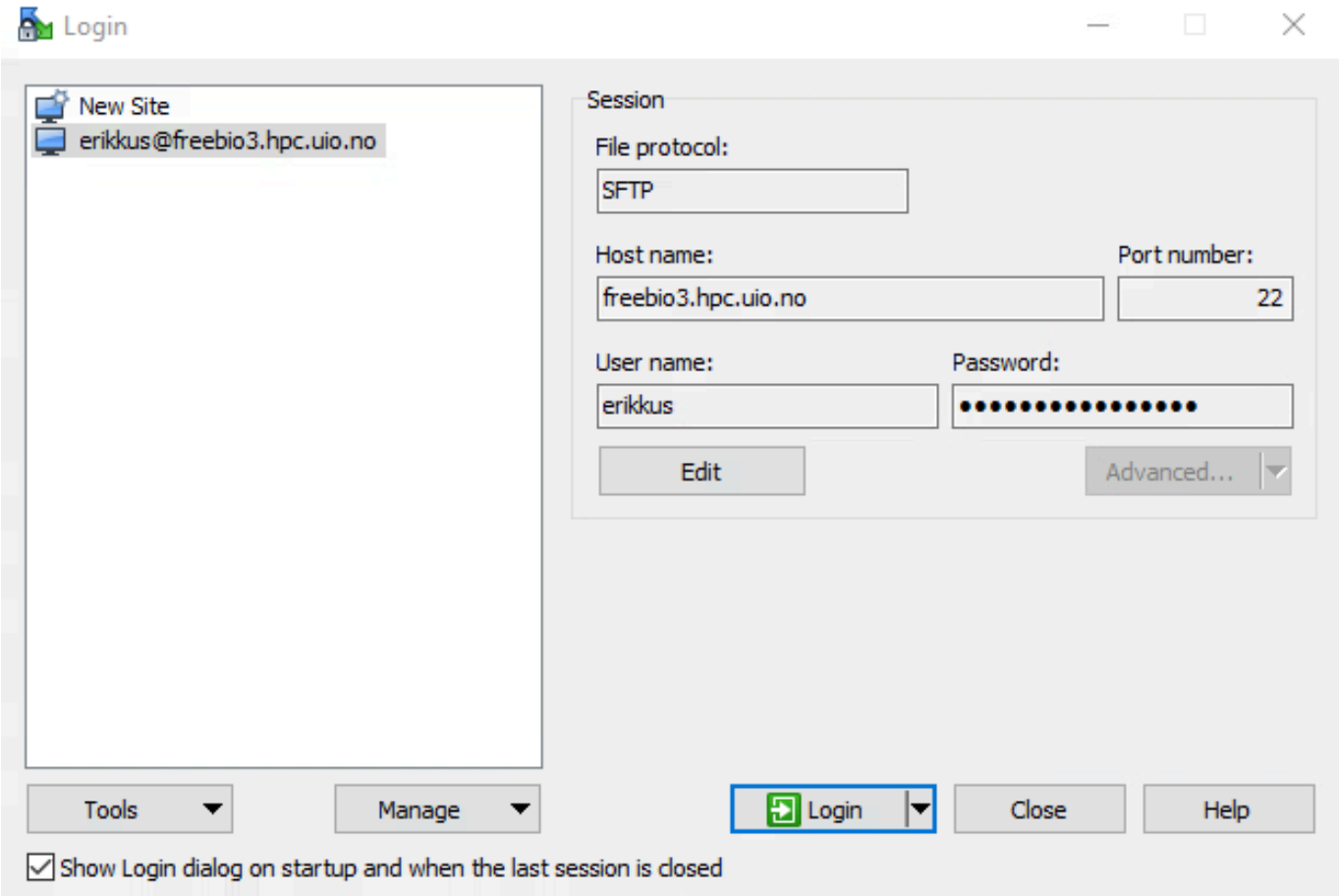
```
erikkus@freebio3:~$ screen -r 3922583.pts-3.freebio3
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=02m 44s
## Animals ##
IS (% of IS required for continued existence) = 0.35
Rewiring Cutoff (probability of rewiring required) = 0.8
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=01h 31m 16s
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=03m 36s
```

Note that the progress bars here are console outputs of one of my scripts.

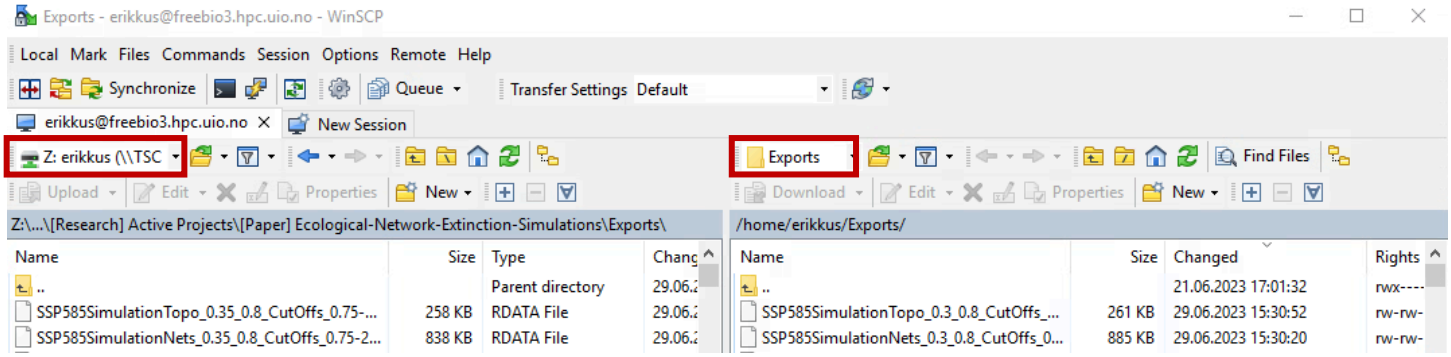
Using this screen procedure, you can disconnect from the server when not in a screen environment and code execution in that screen environment will continue. You can check that you are properly disconnected from each screen when the screen -ls command returns only (Detached) screens.

File Transfer

To inject files into the HPC system and harvest them from there, you can use WinSCP. There are more straightforward solutions available, but I have found this one to be the most user-friendly due to its reliance on a graphical user interface. Just like PuTTY (which we use for code execution in the HPC environment), WinSCP can be started via the VMWare Horizon client or the webbrowser. There, create a "Site" and enter the same host name as you did for your PuTTY session:



You may also enter your username and password if you want them saved for easier re-connections. After you click "Login", the connection is established and you are greeted with a two-pane window – the right-hand side is the server drive, the left-hand side is the windows statistic drive by default. You can now drag and drop between these drives. Alternatively, you can also drag and drop into any of these panes from your local machine. If you wish to change either drives and directories, you can do so via the respective dropdown boxes:



LUMI Supercomputer

LUMI (Large Unified Modern Infrastructure) is an international HPC infrastructure which far eclipses the other computational resources listed in this guide. You can read more about it here.

This is a resource predominantly for BioDT staff at NHM. But Norwegian academic staff can apply for LUMI here. This guide assumes that you work on a project that has already been registered and set up with LUMI.

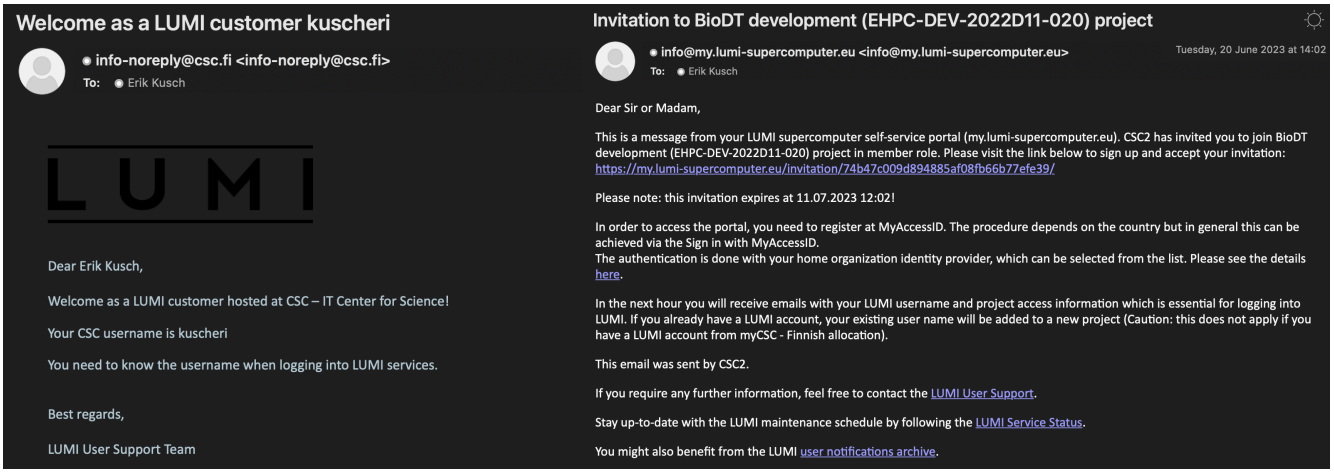
Registration & Getting Access

To register a user account (not a project!) with LUMI, you may follow the guide provided by LUMI directly or my description of the necessary steps listed here:

1. **Create an account.** You will need a Puhuri account. This can be set up following either the Puhuri documentation or by navigating to https://my.lumi-supercomputer.eu/login/ and choosing the "add another institution" option, then selecting Feide (our MyAccessID provider)

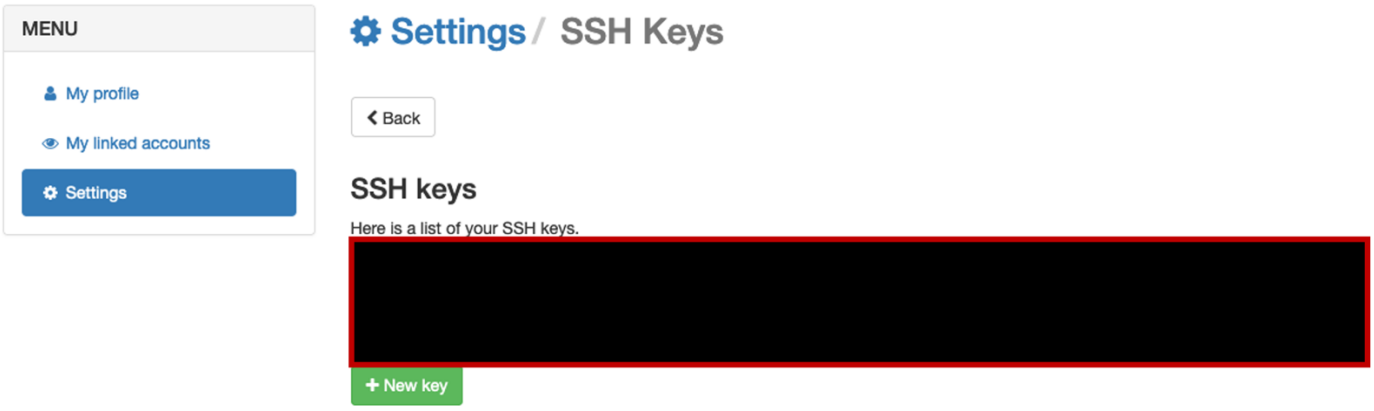
   You will then be prompted to log in with your UiO credentials. Do so.

2. **Project and Username.** Next, you will receive two Emails (see below) prompting you to accept an invitation to the project you are affiliated with (you may have to ask for this invitation to be sent to you). Thereafter, you will receive an E-mail with your username for the LUMI services.



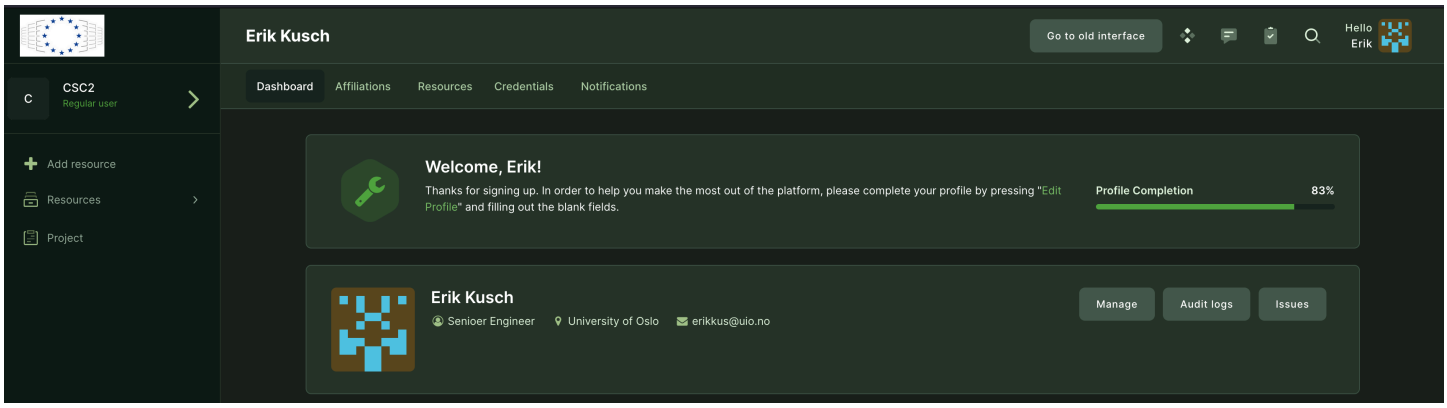
3. **SSH Key.** To log into LUMI itself, you will need to have an SSH key set up. To do so, you may follow [this documentation](). First, open a terminal or run environment, depending on whether you are working on MAC or PC and then run ssh-keygen -t ed25519 This will generate a public and a private SSH key. Alternatively, you can also create such a key with PuTTY or other SSH manager GUIs (see the documentation linked above). During SSH key creation, you will be prompted for a **passphrase – be sure to remember it**!

   Once you have your SSH Key, log on to https://mms.myaccessid.org/profile/, navigate to settings and enter your public key you just created via the "+ New key" button:

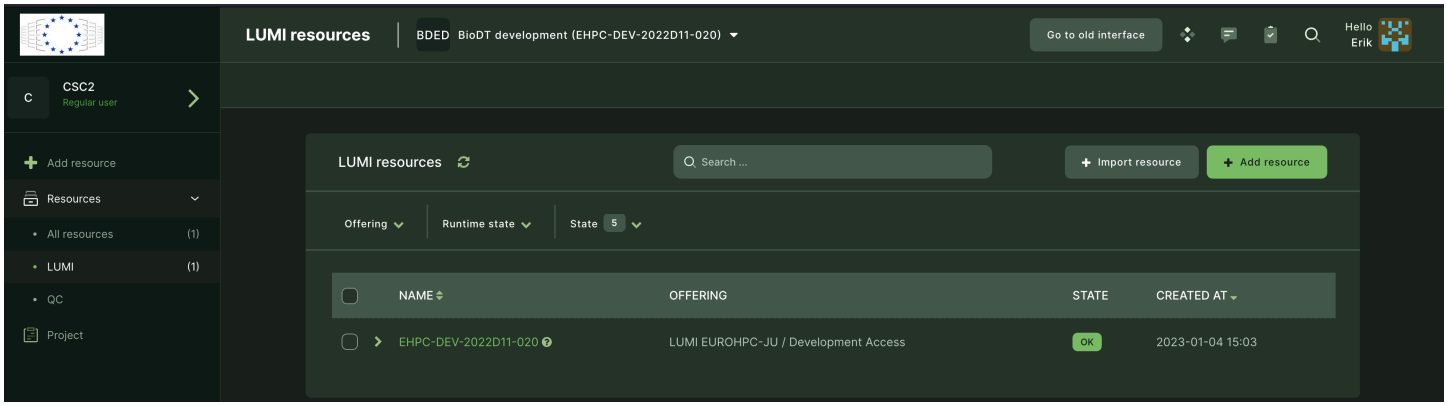


   Finally, before you log onto LUMI, allow for some migration time of your key in the system (roughly an hour).

4. **CSC Portal**. To monitor your allocations for LUMI, you can log onto: https://my.lumi-supercomputer.eu/login/ which should result in a landing page like this:



There, under the resources tab, you should see a listing for your specific project like so for BioDT:



Connecting to the System

To connect to the LUMI HPC itself you may follow this documentation. In general, however, these steps should do:

1. ssh YOURUSERNAME@lumi.csc.fi. This connects you to the server itself. You will be prompted for your passphrase you used when creating your SSH key (see above). When typing your passphrase, you won't see it appear as you type. Hit RETURN when typed to completion:

```
erikkus — ssh kuscheri@lumi.csc.fi — 80×24
Last login: Sun Jul  2 14:06:50 on ttys000
erikkus@Eriks-MacBook-Pro ~ % ssh kuscheri@lumi.csc.fi
Enter passphrase for key '/Users/erikkus/.ssh/id_ed25519':
```

   This will log you onto LUMI itself:

```
The Supercomputer of the North
```

2. Next, you will want to check your quota and project allocation with: lumi-workspaces (this will only be necessary to do when logging in for the first time or attempting to start a new project on the HPC):

```
kuscheri@uan02:~> lumi-workspaces

Quota for your projects:

Disk area                          Capacity(used/max)  Files(used/max)
-----------------------------------------------------------------------
Personal home folder
Home folder is hosted on lustrep3

/users/kuscheri                             15M/22G        113/100K
-----------------------------------------------------------------------
Project: project_465000357
Project is hosted on lustrep2

/projappl/project_465000357                5.9G/54G         33K/100K
/scratch/project_465000357                 2.3T/55T         71K/2.0M
/flash/project_465000357                   4.1K/2.2T          1/1.0M
-----------------------------------------------------------------------

Status of your allocations:

Data updated: 2023-07-03 16:16:04
Project            |                CPU (used/allocated)|              GPU (used/allocated)|          Storage (used/allocated)
-----------------------------------------------------------------------------------------------------------------------------------
project_465000357  |       23639/1920000   (1.2%) core/hours|            0/0    (N/A) gpu/hours|       1872/90000   (2.1%) TB/hours
```

   In our case, BioDT here is set as project_465000357.

3. Enter the working directory of your project and create a directory for your specific work (so as to not interfere with others):

   cd /scratch/YOURPROJECTNUMBER
   mkdir YOURUSERNAME
   cd YOURUSERNAME

   Once done, your terminal line should look similar to this:

```
kuscheri@uan02:/scratch/project_465000357/kuscheri>
```

   This is your personal space – here you can store data and run code as you please on LUMI.

Coding & Code Execution

Loading Software & Modules

To get started running code on LUMI, we first need to start the code environment (R, in our case). To do so, we need to load the program itself into our personal HPC environment. This environment gets created as empty when we log on and we now want to load the R program into it. This is how new do it:

1. **Identify which version of the program is available to us.** This is done with the terminal command:
   module spider cray-R
   Modules are what the HPC environment calls programs. Spider indicates for the HPC to look for something matching the writing following the word spider. The output we receive is:

```
kuscheri@uan02:/scratch/project_465000357/kuscheri> module spider cray-R

----------------------------------------------------------------------------
  cray-R:
----------------------------------------------------------------------------
     Versions:
        cray-R/4.1.3.1
        cray-R/4.2.1.1

----------------------------------------------------------------------------
  For detailed information about a specific "cray-R" package (including how to load the modules) use the module's full name.
  Note that names that have a trailing (E) are extensions provided by other modules.
  For example:

     $ module spider cray-R/4.2.1.1
----------------------------------------------------------------------------
kuscheri@uan02:/scratch/project_465000357/kuscheri>
```

   Swapping out cray-R (cray is the provider of R on LUMI) for something else, we can find other modules.

2. **Loading the program into our HPC Environment.** This is like loading libraries in R itself in that we make the module itself available to the HPC environment. We do this via the command: module load cray-R/4.2.1.1
   In the future, you may need to change the R version according to which versions you obtain with step 1 above:

```
kuscheri@uan02:/scratch/project_465000357/kuscheri> module load cray-R/4.2.1.1
```

3. **Prepare library folder for R packages.** You need to prepare a local folder for R package installation. You can do so before opening the R environment by running the following two lines in the HPC terminal:
   export R_LIBS=~/local/rlibs
   mkdir -p ~/local/rlibs

```
kuscheri@uan02:/scratch/project_465000357/kuscheri> export R_LIBS=~/local/rlibs
kuscheri@uan02:/scratch/project_465000357/kuscheri> mkdir -p ~/local/rlibs
```

4. **Start the module/R.** Now we can start the R environment via the terminal command: R
   Now, you find yourself in an R environment in which you can code like normal:

```
kuscheri@uan02:/scratch/project_465000357/kuscheri> R

R version 4.2.1 (2022-06-23) -- "Funny-Looking Kid"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-suse-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

During startup - Warning message:
Setting LC_CTYPE failed, using "C"
```

Interactive Coding

Following the procedure above, you find yourself in the familiar R console environment. Here you can code just like you would in an R console instance on your local machine. Some useful commands you should be aware of in this environment are:

`getwd()` – shows current working directory

`list.files()` – lists all files in current working directory

`source()` – executes a code file

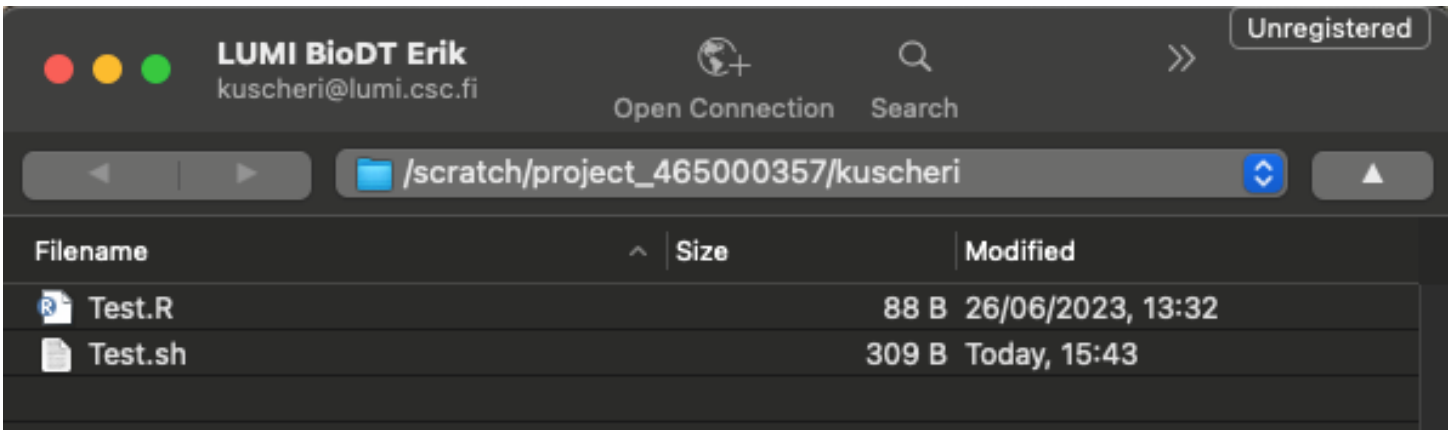
`q()` – quits the R environment

Note that **any code run in this environment will terminate execution when you disconnect from the server**. To avoid this, you need to go through **unsupervised code execution**!

Unsupervised Code Execution

To run R code on LUMI, we must provide the R script, associated data, and a shell script that tells LUMI how to run it.

When looking at the directory in which we want to execute code, we thus need at least two files (more on how to see the directories on LUMI like this in the section on file transfer further down). In my demonstration case, the R script is called Test.R and the shell file is called Test.sh:



When creating the relevant R script and shell script, it is good to consider the following:

The R Script

Make sure your script:

- Is self-contained
- Has soft-coded working directories!
- Installs non-installed packages before loading them

The Shell Script

This is a .sh file. Effectively, it tells the cluster how to treat our job. It looks something like this:

Test.sh

```
#!/bin/bash
#SBATCH --account=project_465000357
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --mem-per-cpu=8G
#SBATCH --time=00:10:00
#SBATCH --partition=small
#SBATCH --job-name=EKRtest
#SBATCH --mail-type=ALL
#SBATCH --mail-user=erik.kusch@nhm.uio.no

module load cray-R/4.2.1.1

Rscript Test.R
```

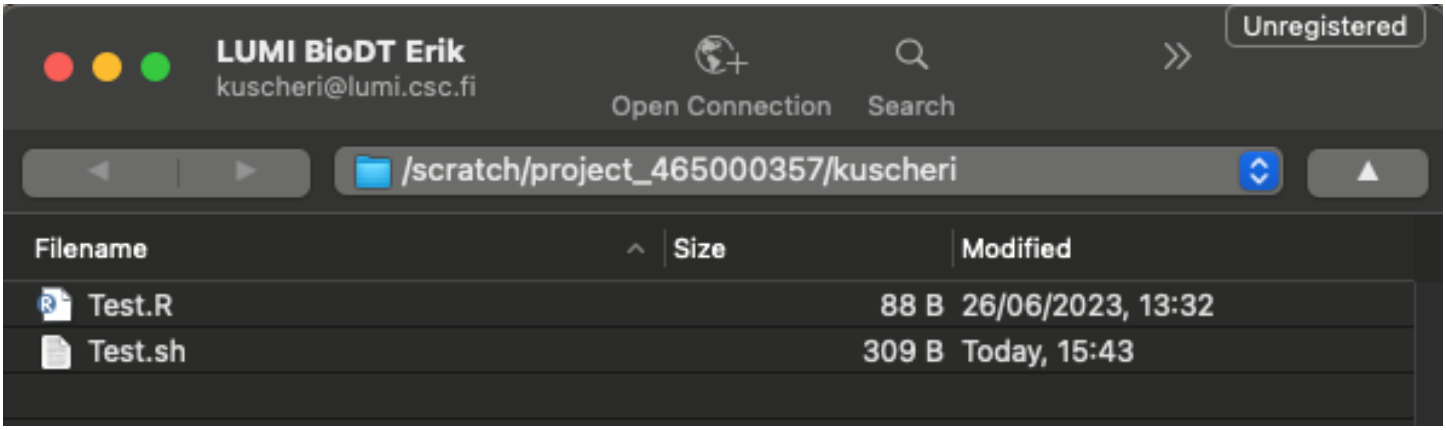
These lines come together as follows:

```
#!/bin/bash
#SBATCH --account=project_465000357
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --mem-per-cpu=8G
#SBATCH --time=00:10:00
#SBATCH --partition=small
#SBATCH --job-name=EKRtest
#SBATCH --mail-type=ALL
#SBATCH --mail-user=erik.kusch@nhm.uio.no

module load cray-R/4.2.1.1

Rscript Test.R
```

Note than I am not registering a local library for R packages as I have previously done so on my first log on to LUMI and I recommend you do the same.

Submitting the job

Make sure your server-directory contains your R script, submission shell script, and all relevant data. You can do so by checking the directory using Cyberduck (see file transfer). It should look something like this in WinSCP:



➔ I don't need any external data for my test script. Hence why there are no data files in this directory.

In the console (terminal or PuTTY):

1. ssh into the system
2. Step into the project directory:

   cd YOURDIRECTORY

```
kuscheri@uan04:~> cd /scratch/project_465000357/kuscheri/
kuscheri@uan04:/scratch/project_465000357/kuscheri>
```

3. Submit the job by calling the shell script (described above):

   sbatch Test.sh

```
kuscheri@uan04:/scratch/project_465000357/kuscheri> sbatch Test.sh
Submitted batch job 3861329
```

Monitoring the Job

1. Find all currently registered jobs for you (this includes any active interactive connections):

   squeue -u YOURUSERNAME

```
kuscheri@uan04:/scratch/project_465000357/kuscheri> squeue -u kuscheri
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           3861329     small  EKRtest kuscheri  R       0:02      1 nid002470
kuscheri@uan04:/scratch/project_465000357/kuscheri>
```

   ➔ The important information here is the JOBID. In the following, it is written as 12345678.

2. Check the progress by showing you the console output of your R script:

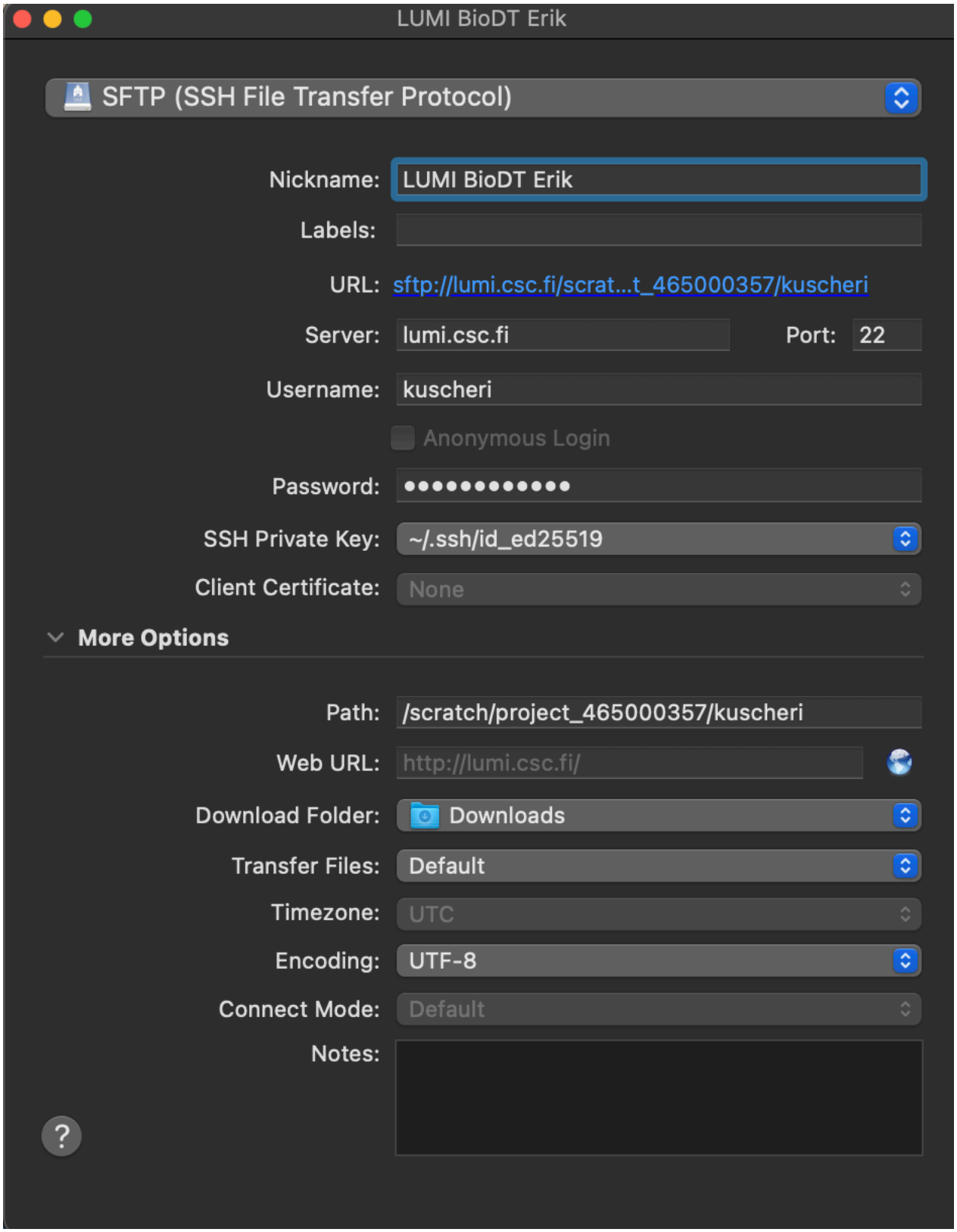
   cat slurm-12345678.out
   → Snapshot of the console.
   tail -f slurm-12345678.out
   → Live updates of the console.

3. Cancel a job:

   scancel 12345678

File Transfer

To connect to LUMI and exchange files, we can use any SFTP manager you prefer. Personally, I use Cyberduck. To set up a direct connection to my personal directory in the BioDT project, I specify Cyberduck as follows (notice the inclusion of the Path and the private SSH key):



Useful Commands

R-Environment

getwd() – shows current working directory

list.files() – lists all files in current working directory

q() – quits the R environment

Unix & Screen Environments

The cluster console environment is a linux environment.

TAB – tries to autocomplete the currently typed word (this is a button-press)

cd XYZ – sets current directory to folder XYZ (contained in current directory), supports absolute paths

cd .. – jump up one directory level

ls – lists all files in current working directory

right-click – pastes (1) from your clipboard or (2) highlighted text in the terminal

module load XZY – loads linux module XZY

module spider XYZ – searches the cluster for available versions of module XYZ

cat XYZ – prints contents of file XYZ

Ctrl+c – force-stops currently executed code. This is a keyboard shortcut.

Ctrl+d – logout command. Will lock you out off ssh and srun sessions. This is a keyboard shortcut.

screen – screen opening command. Will create a screen environment. This is a terminal line, write it out, then hit RETURN.

Ctrl+a+d – detach screen command. Will close a screen environment but continue execution of code therein. This is a keyboard shortcut.

screen -list – screen indexing command. Will return a list of screen environments you have closed. This is a terminal line, write it out, then hit RETURN.

screen -r XXXX – screen re-opening command. Will reopen the screen indexed as XXXX (indexing as according to what is returned by screen -list). This is a terminal line, write it out, then hit RETURN.

Ctrl+a+k – kill screen command. Will close a screen environment and terminate all code execution therein. This is a keyboard shortcut.

screen -X -S XXX quit – kill detached screen command. Will kill a screen that is detached. This is a terminal line, write it out, then hit RETURN.