

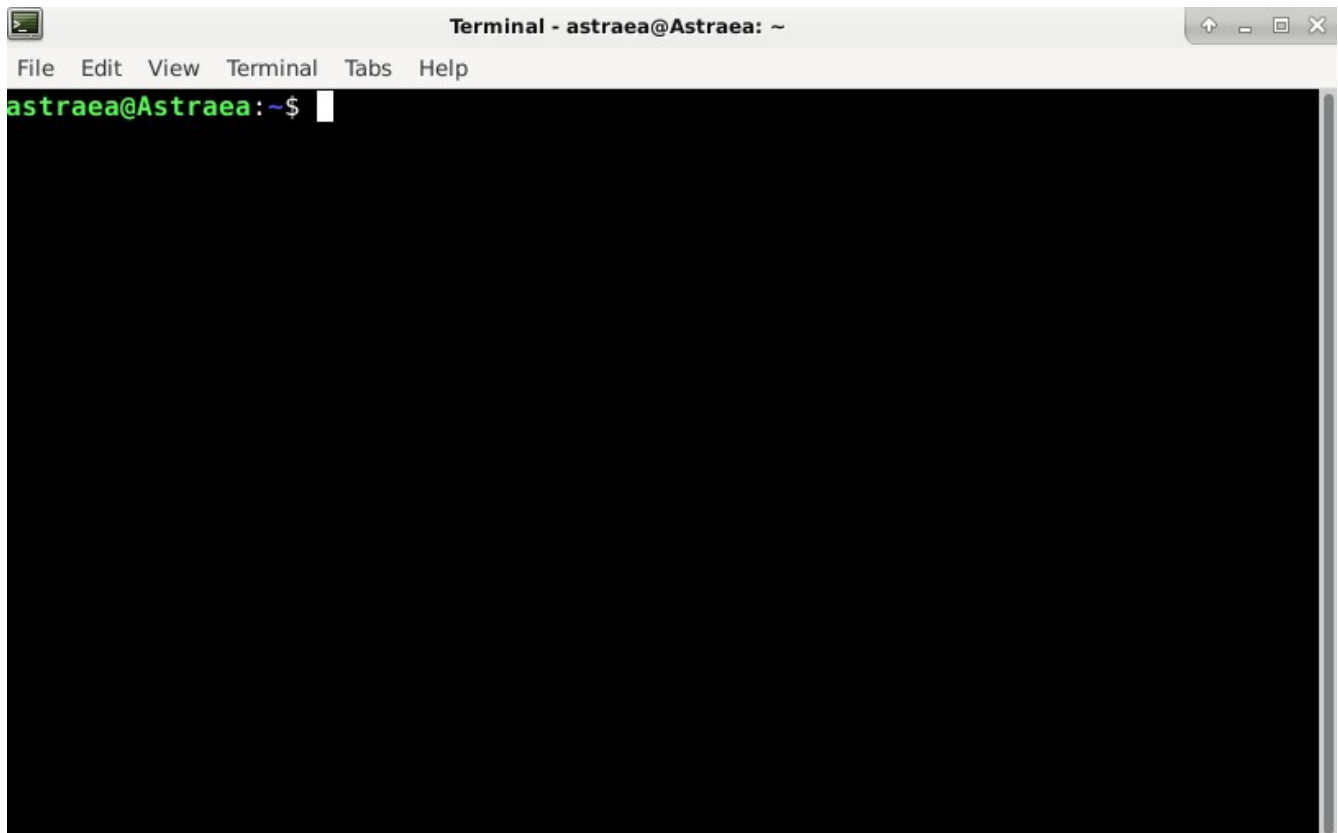
General guide to the Radboud computer cluster

Written by Astraea Blonk, June 19 2021

If researchers and students at Radboud University want to run large or heavy scripts that can't be handled by their own PCs, they are allowed to access the Radboud computer cluster remotely, upload their script there and tell one of those computers: "please run this script for me". I will explain each of the steps necessary to do so in this document.

First, an important thing to note is that these computers have no graphical interface (as far as I know). This means we'll have to resort to the **Command Prompt/cmd (Windows)** or **Terminal (Mac/Linux)**. You should be able to open the Command Prompt in Windows 10 by using the search bar in the lower left corner on the screen. A guide for opening the Terminal on Mac can be found [here](#).

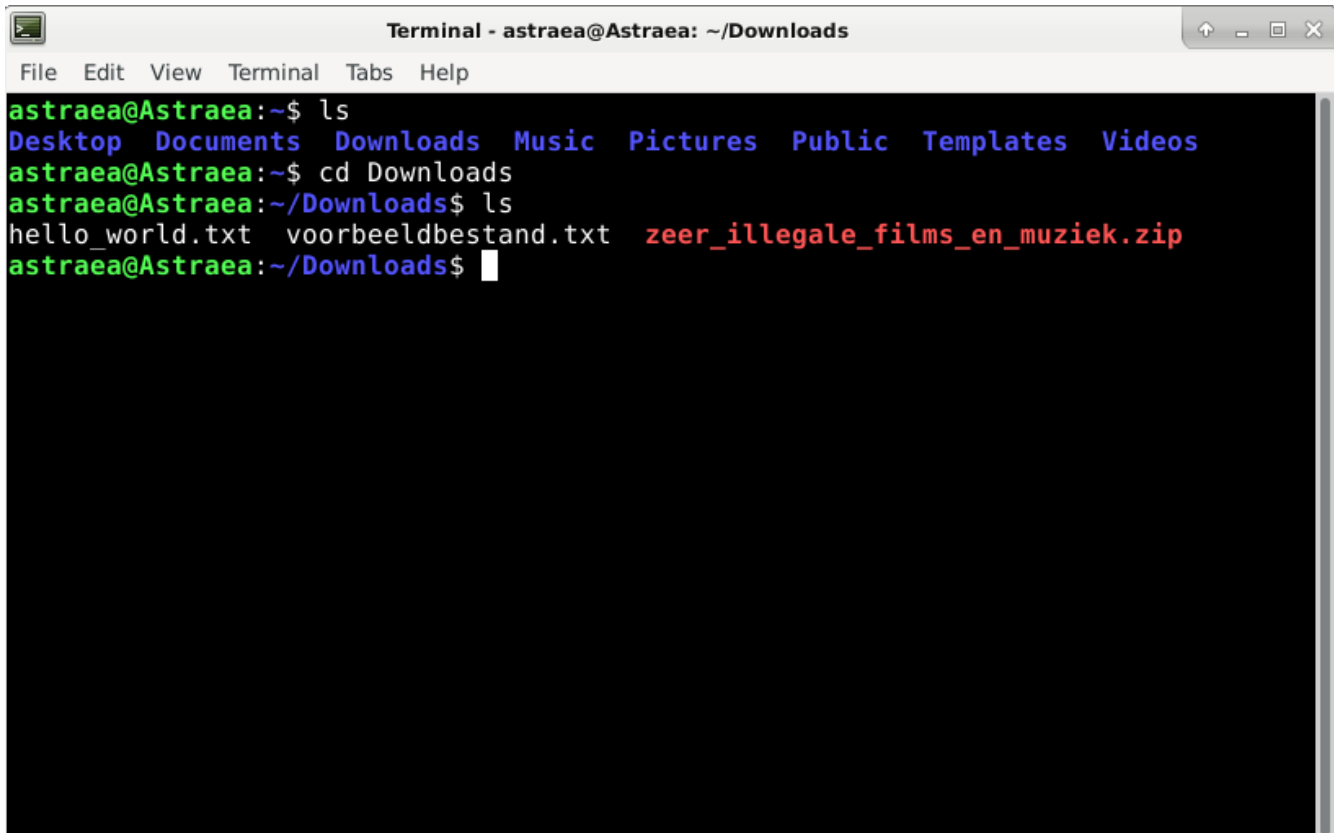
The Command Prompt and the Terminal are examples of **Command Line Interfaces (CLIs)**. For those who have never worked with a CLI before; this may sound like a name for something really complex, but the concept is actually fairly simple if you know how a regular computer works (which you should if you're reading this): instead of a "graphical interface" - where you have your background with icons that you can click on to start programs - we now have a "command line interface": a black screen in which we start programs by directly typing "commands". No mouse clicks needed.



Picture 1: an example of what a CLI might look like (on Linux)

You can do the same things with a CLI as with a regular graphical interface. For example, during normal computer usage we might want to open Windows Explorer (or a similar Mac/Linux program) and look through our files. We can do the same thing on a CLI. Try, for example, typing **ls** (and then pressing <enter>). This command name is short for "list" and tells your pc to show a list of file names

and directory names in the current directory. By default, the CLI starts in your home directory, so the `ls` command shows a list of files in your home directory, but you can change to a different directory by typing `cd`, followed by the **name of directory you want to go to**. For instance, typing `cd Downloads` should bring us to the Downloads folder (if this doesn't work because you don't have a Downloads folder, try another directory from the list of files shown by `ls`). If this was successful, you can type in the command `ls` again, and it should show a different list of file names now! Woooow, magic...

A terminal window titled "Terminal - astraee@Astraee: ~/Downloads". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal content shows the following sequence of commands and outputs:

```
astraea@Astraee:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
astraea@Astraee:~$ cd Downloads
astraea@Astraee:~/Downloads$ ls
hello_world.txt  voorbeeldbestand.txt  zeer_illegale_films_en_muziek.zip
astraea@Astraee:~/Downloads$
```

Picture 2: what a CLI could look like after using "ls" and "cd". If you use this command yourself, you should see the names of your own files, instead of the ones on my pc.

With the basics of using a CLI out of the way, we can turn to our specific use case: accessing and running files on the Radboud computer cluster. There are more useful commands other than just `ls` and `cd`, but I will talk about those later in this guide, when each of them becomes relevant. I will also be putting a table at the end of this document with a list of all commands used in this guide and a short description of how to use them.

Anyway, when you used `cd` to go to a different directory, you might have noticed that the thingy on the left side of the screen, which is shown just left to the place where you can type in your commands, has changed. This thing is called a **prompt** and it tells you some basic information about who is using the CLI and what directory you are in. We will see that the prompt will also change when we start accessing the Radboud computer cluster. This is because, in order to upload/download files and run scripts on one of the Radboud computers, we have to make a **remote connection** to it, and once we have created that connection, we can type commands like `ls` and `cd` on the remote computer as if we were typing them on our own computer. You will see what I mean in a second. First, let's create the connection. This can be done using the `ssh` command, followed by the **name of the computer that you want to access**. For this guide, I will assume that you want to use the computer

blossomforth.science.ru.nl¹. However, trying to type `ssh blossomforth.science.ru.nl` will give you an error. This is because of the way the Radboud computer cluster is set up: we have to log in through a **login server** first, before we can create a connection to either `blossomforth.science.ru.nl`. To connect to the login server, we can type `ssh <username>@applejack.science.ru.nl`, where `<username>` is to be replaced by your personal science login username. For example, if I personally want to login, I would type `ssh ablonk@applejack.science.ru.nl`, but for you the command should look slightly differently, depending on the username of your science login credentials.

When you've successfully used the command I described above using your own username, you may get a complicated sounding message about authenticating the remote host – just type in “yes” if you get that, it should only appear the first time you try to log in – followed by a “password:” prompt. This is when we type in the login password associated with our science login. **NB: the CLI will not provide any indication that you are typing your password.** No dots or anything. This is okay, you can just type your password without the screen changing, hit `<enter>`, and everything should work (that is, if you put in the right password).

A quick side note, feel free to skip this: if you realize you've made a mistake while typing your password but have already pressed the `<enter>` button, you would normally have to wait for the login verification process to fail before you can try out your actual password. However, you can always terminate (stop) ongoing commands that a computer is executing by using `<CTRL>+C` or **COMMAND+C**. This will abort the login verification process and allow you to type new commands. You can then input the `ssh` command again to get a new try at typing your password. You can also just wait for the login verification process to fail and it should give you a new chance to put in your password as well.



Picture 3: logging in to the Radboud login server Applejack.

When you've entered a correct combination of science login username and password, you will enter a connection with the Radboud login server, “applejack.science.ru.nl”. You will know this to have worked when you see a picture of Applejack from My Little Pony. Congratulations, you've made a connection to a remote computer, and can give it commands just like you would be able to on your own PC! However, Radboud has a lot of different computers, and this particular computer is just a login server, and not the computer that was actually designated to us to do heavy computations. For that, we

¹ See also <https://ponyland.science.ru.nl/doku.php?id=wiki:ponyland:about> for a list of available computers.


```

[ablonk@blossomforth:~]$ cd /vol/tensusers5/
[ablonk@blossomforth:/vol/tensusers5]$ ls
akhan      ctejedor  disk_usage  ihendrickx  Lissabon  ripd      vmeijer  xwei
arifkhan   cvoeten   ekomen      jjzurne     mbentum   swills    wstoop
[ablonk@blossomforth:/vol/tensusers5]$ mkdir ablonk
[ablonk@blossomforth:/vol/tensusers5]$ ls
ablonk  arifkhan  cvoeten   ekomen      jjzurne  mbentum   swills    wstoop
akhan   ctejedor  disk_usage  ihendrickx  Lissabon  ripd      vmeijer  xwei
[ablonk@blossomforth:/vol/tensusers5]$

```

Picture 5: Creating your own directory on the server

All nice and well, but how do we upload our scripts to this new directory? The good news is: there is a command to do that called **scp**! The bad news is: we can't use it while we're connected to the Radboud computer. We'll have to log out and get back to our own PC first. To abort the remote connection, type **exit** twice. The first **exit** should bring you back to the applejack.science.ru.nl computer (which we used to login), and the second **exit** disconnects you from the login server and brings you back to the place where you can issue commands to your own computer. From here, we can upload scripts to the Radboud computers by typing **scp <path-to-script-on-pc>**

<username>@applejack.science.ru.nl:<path-to-upload-directory>. This may look complicated, so I'll give an example: if I want to upload a script called "test.py" which is saved in directory "C:\Users\Astraea\Research\" and upload that to my personal directory on the Radboud computer ("/vol/tensusers5/ablonk/"), the command that I'd use would look like this: **scp C:\Users\Astraea\Research\test.py ablonk@applejack.science.ru.nl:/vol/tensusers5/ablonk/**. If you ever need to do this yourself, just copy the template that I described above and make sure to **replace (1) <path-to-script-on-pc>**, **(2) <username>** and **(3) <path-to-upload-directory>** with **(1) the path to the file you want to upload**, **(2) your science login username**, and **(3) the path to your directory on the Radboud computer, respectively**.

Running the above command or a similar command will ask me to enter my password (twice), and possibly, to authenticate the remote host, just like with the **ssh** command. Under the hood, **scp** actually creates a temporary **ssh** connection to send the files from your local PC to the Radboud computer, but that's some technical detail that you don't need to remember. When you finish typing in your password, the CLI will say that it's uploading the file, along with a percentage. This percentage should reach 100% pretty quickly (if your file isn't too large), and when it does, you'll get your prompt back and you can continue your command typing journey.

```

astraea@Astraea:~$ scp /home/astraea/test.py ablonk@applejack.science.ru.nl:/vol/
/tensusers5/ablonk/
ablonk@applejack.science.ru.nl's password:
test.py                               100%  21      0.2KB/s   00:00
astraea@Astraea:~$

```

Picture 6: Uploading your own file to the server

Now that we've uploaded our script, it's time to tell the Radboud computer to run it. First, we need to log in to applejack.science.ru.nl and switch to blossomoforth.science.ru.nl using the **ssh** command that I described earlier. You can check if your script has been uploaded by changing to your personal directory and typing **ls**, which should now show that your directory contains one file called "test.py" (or whatever the name of your own script was).

Before we can run this script, we should check if the computer is busy using the command **htop**. Using this command will open an overview of what the computer is doing; the upper half will show a list of 32 processing components (called "cores") and how busy they are, and the lower half will show a list

of specific commands that the computer is currently trying to execute, along with some info about the user who typed in those commands, etc. This overview may look like it gives you a frightening amount of information at once, but it will suffice here to look only at the percentages shown in the upper half. If you see a lot of 0%, it's okay to use this computer. If all the percentages are close to 100%, someone else is probably running their own heavy script already and it might be better to switch to another Radboud computer.

```

Terminal - astraee@Astraea: ~
File Edit View Terminal Tabs Help

 1 [ 0.0%] 9 [ 0.0%] 17 [ 0.0%] 25 [ 0.0%]
 2 [ 0.0%] 10 [ 0.0%] 18 [ 0.0%] 26 [ 0.0%]
 3 [ 0.0%] 11 [ 0.0%] 19 [ 0.0%] 27 [ 0.0%]
 4 [ 0.0%] 12 [ 0.0%] 20 [ 0.0%] 28 [ 0.0%]
 5 [ 0.0%] 13 [ 0.0%] 21 [ 0.0%] 29 [ 0.0%]
 6 [ 0.0%] 14 [ 0.0%] 22 [ 0.0%] 30 [ 0.0%]
 7 [ 0.0%] 15 [ 0.0%] 23 [ 0.0%] 31 [ 0.0%]
 8 [ 0.0%] 16 [ 0.0%] 24 [ 0.0%] 32 [ 0.0%]
Mem [|||||] 39.4G/251G Tasks: 123, 161 thr; 1 running
Swp [||] 57.5M/25.6G Load average: 0.01 0.00 0.00
Uptime: 95 days, 22:44:41

  PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
 1885          20   0 153M  2684 2480  S   0.7  0.0  1:02.61
 3485          20   0 672M 170M 6384  S   0.7  0.1 18:40.10
 2189          20   0 672M 170M 6384  S   0.7  0.1 58:27.37
18255          20   0 28732 4156 3508  R   0.0  0.0  0:00.08
 3461          20   0 998M 72612 32820 S   0.0  0.0 1h31:53
 1668          20   0 108M  3160  2732 S   0.0  0.0 12:24.31
 2401          20   0 998M 72612 32820 S   0.0  0.0 1h55:13
    1          20   0 220M  9972  6932 S   0.0  0.0  5:37.03
   824          20   0 23604 4196  3492 S   0.0  0.0  0:00.94
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit

```

Picture 7: What using "htop" might look like

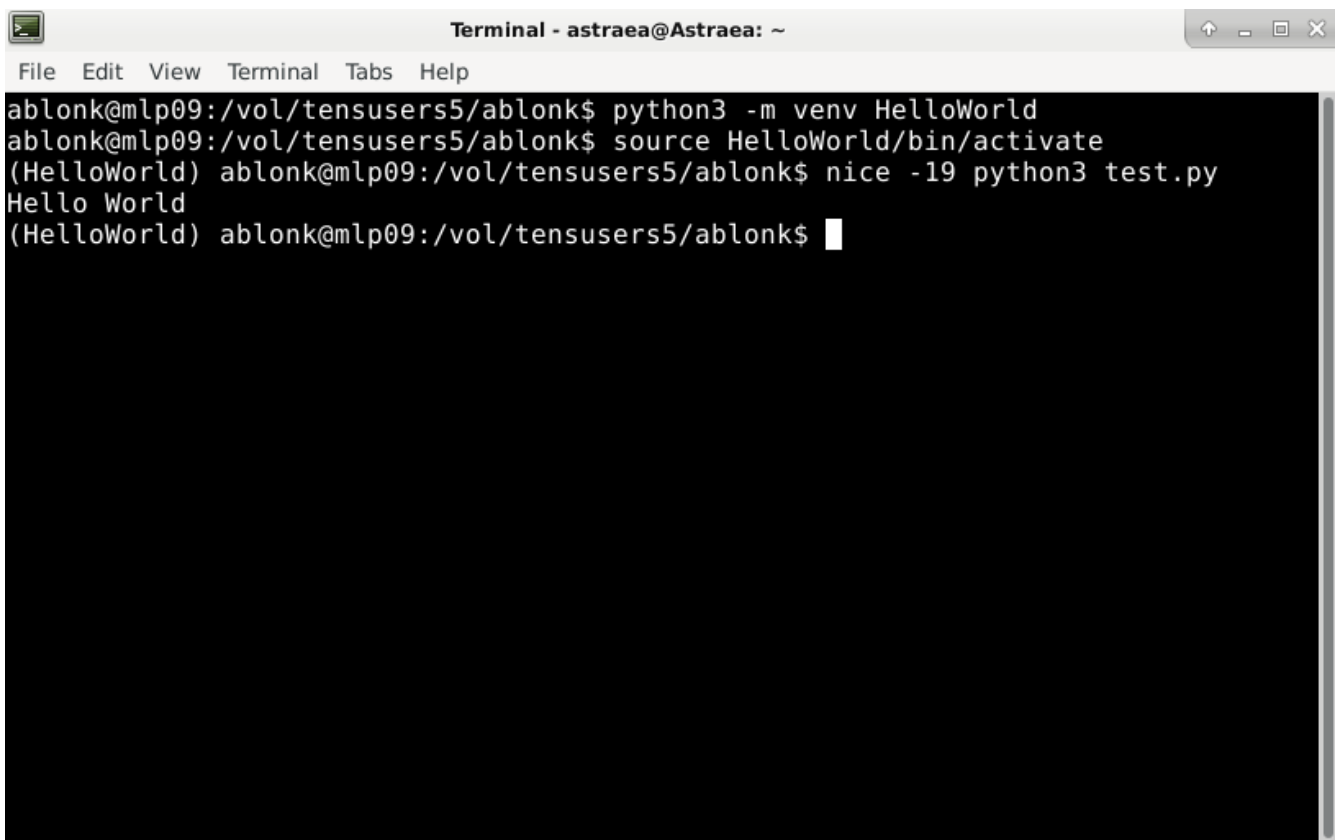
When you've found a computer with a reasonable amount of unused or barely used cores, you're almost set to launch your script. We might just type in `python3 <path-to-script>` to run our script already, and this is fine if you're planning to run a simple script, but when dealing with more complicated/heavy scripts, running this command straight away may result in some trouble. Let me just give you 3 extra bits of advice which may come in handy when handling longer scripts:

1. Be **nice**. Or rather, use **nice**. This command tells the Radboud computer that the script we want to run has a low priority, and that if there's anybody who is in a hurry to run something else they can use a command with a higher priority to temporarily take away your cores and return them when their script has finished. If you wanna run a script that you expect will be going on for multiple hours, I really suggest you use **nice** to prevent other computer users from getting mad at you because you've taken up all the processing power without them being able to run a short but urgent script in between :). To run a script the 'nice' way, type **nice -19 python3 <path-to-script>**, with <path-to-script> being something like `"/vol/tensusers5/ablonk/test.py"`.
2. Use a **screen**. Normally, when you close the CLI, it disconnects from the Radboud computer and aborts whatever command or script you were trying to run. If you have a script that's going to be running for a while, it might be annoying to have to keep your PC turned on and the CLI window open. To make sure we can turn off our computer at night and increase our chances of getting quality sleep time, we can create a **virtual "screen"** using the **screen** command. This

will open an empty window where we can type in commands. Here, if we type `nice -19 python3 <path-to-script>` and then press `<CTRL>+D` or `COMMAND+D`, we will “detach” from our virtual screen and return to the place where we typed in the `screen` command. We can then use `exit` to disconnect from the Radboud computer and our virtual screen will make sure our command keeps running while we’re gone. To re-attach to a virtual screen (for example, to check if your script is working as intended or to see whether it’s finished already or not), connect with the computer where you originally created your screen session and type `screen -r`, with “-r” meaning “resume”. This will reopen the virtual screen in which you started your script. You can then detach from it again using `<CTRL>+D/COMMAND+D` or, if the script has finished, you can type `exit` to destroy the screen.

3. Use **virtual environments**. Sometimes, when we want to run a script, the CLI will give us an error message saying that it “can’t find a module called ‘XXX’”, where XXX is the name of a specific Python module that your script depends on. We *could*, of course, run `pip install XXX` straight away to install the necessary module on the Radboud computer. However, the number of different Python libraries in existence is so enormous that if everyone did what I just described, the Radboud computers would soon be chock full of niche Python modules and there would be no room for anyone to save their script or their output anymore. That’s why, if you need access to a module that’s not yet installed, you are advised to create a temporary “virtual environment” and install the module in there. When you’re done using your virtual environment, it can be deleted, along with all of the modules that were installed in there, freeing up disk space for others to use.

To create a virtual environment, type `python3 -m venv <environment-name>`. `<environment-name>` can be entirely random if you like, but I personally tend to make sure it refers at least slightly to the project that I’m using it for or the module that I want to install in there. After creating the virtual environment, we can activate (use) it by typing `source <environment-name>/bin/activate`. We can then proceed to install those python modules that we got an error about using pip.

A terminal window titled "Terminal - astraea@Astraea: ~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal content shows a sequence of commands and their outputs:

```
ablonk@mlp09:/vol/tensusers5/ablonk$ python3 -m venv HelloWorld
ablonk@mlp09:/vol/tensusers5/ablonk$ source HelloWorld/bin/activate
(HelloWorld) ablonk@mlp09:/vol/tensusers5/ablonk$ nice -19 python3 test.py
Hello World
(HelloWorld) ablonk@mlp09:/vol/tensusers5/ablonk$
```

Picture 8: Executing a Python script on a screen, from a virtual environment, the nice way

And... that's how you run a script on the Radboud computer cluster! I hope the most important aspects and commands associated with this process have been made clear to you. If not, you can send an e-mail to admin@cls.ru.nl or consult the [wiki](#). If you have a question about how to use one of the specific commands named in this document, you can also consult the **list of commands** on the final two pages of this document.

Oh, two more things: of course, there's gonna be a point in time where your script has finished and has generated some output files. First, to download those output files, just use the `scp` command but with the order of the last two arguments reversed: instead of using it like `scp <path-to-script-on-pc> <username>@applejack.science.ru.nl:<path-to-upload-directory>`, use it like `scp <username>@applejack.science.ru.nl:<path-to-output-directory> <path-to-project-directory-on-pc>`. The penultimate argument denotes the place where the file should come from and the final argument describes where it should go, so, by playing around with the order, you can use this command to both upload files as well as to download them.

Second, if your script has finished, you've downloaded your output and you think it won't be necessary to keep the script/output on the Radboud computer cluster anymore, you can delete those unneeded files by navigating to their directory and typing `rm <name-of-file>`. This will remove the specified file and free up space for other users.

Have fun scripting!

List of commands

Command	Description	Examples
ls	Shows a list of files and subdirectories that are inside the current directory.	ls
cd <name-of-directory>	Change to a different directory	cd Downloads cd /vol/tensusers5/
ssh <username>@applejack.science.ru.nl	Create a connection to the Radboud computer cluster login server.	ssh ablonk@applejack.science.ru.nl
ssh <name-of-computer>	After logging in to the login server, this command can be used to connect to a different Radboud computer.	ssh blossomforth.science.ru.nl
mkdir <name-of-directory>	Create a new subdirectory inside the current one.	mkdir ablonk
exit	Use this to disconnect from a remote computer, destroy a screen, or close the CLI entirely.	exit
scp <path-to-script-on-pc> <username>@applejack.science.ru.nl:<path-to-upload-directory>	Upload a file (such as a script) to the Radboud computers from your PC.	scp C:\Users\Astraea\Research\test.py ablonk@applejack.science.ru.nl:/vol/tensusers5/ablonk/
htop	Check whether the current computer is busy or not.	htop
python3 <path-to-script>	Execute a Python script, the not so nice way.	python3 /vol/tensusers5/ablonk/test.py
nice -19 python3 <path-to-script>	Execute a Python script, the nice way.	nice -19 python3 /vol/tensusers5/ablonk/test.py
screen	Create a virtual screen (use <CTRL>+D/COMMAND+D to detach).	screen
screen -r	Reattach to a virtual screen.	screen -r
python3 -m venv <environment-name>	Create a virtual environment	python3 -m venv HelloWorld
source <environment-name>/bin/activate	Activate a virtual environment	source HelloWorld/bin/activate
pip install <module-name>	Install a Python module, to be used in a script.	pip install pandas

scp <username>@applejack.science.ru.nl:<path-to-output-directory> <path-to-project-directory-on-pc>	Download a file (such as an output text file) from the Radboud computers to your PC.	scp ablonk@applejack.science.ru.nl:/vol/tensusers5/ablonk/test_results.txt C:\Users\Astraea\Research\
rm <name-of-file>	Delete a file.	rm confusion_about_how_to_use_the_RU_computer_cluster.txt (get it? I explained how to do this in this document!)