

4. Bash & File Management

What is Linux

Linux is an open-source operating system kernel. An operating system kernel is the core component that manages hardware resources and allows higher-level software to run. Linux is known for its stability, security, and versatility. It forms the basis for many different Linux distributions

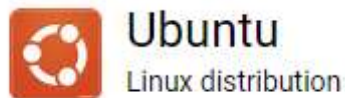
It is widely used for servers, embedded systems, and as an alternative desktop operating system. Different organizations and individuals can build their own Linux distributions based on the Linux kernel, and these distributions vary in terms of included software and user interface.



What is Ubuntu

Ubuntu is a popular Linux distribution based on Debian. It is designed to be user-friendly and comes with a variety of pre-installed software for ease of use. Ubuntu is known for its regular release cycle and long-term support (LTS) versions, which receive updates and support for an extended period.

Ubuntu provides a graphical user interface (GUI) similar to that of other operating systems like Windows or macOS. It's often recommended for users who are new to Linux because of its user-friendly approach and extensive community support.



What is a Terminal

A terminal, also known as a command-line interface (CLI) or shell, is a text-based interface in which users interact with the computer by typing commands. It provides a way to navigate the file system, run programs, and perform various tasks using text commands.

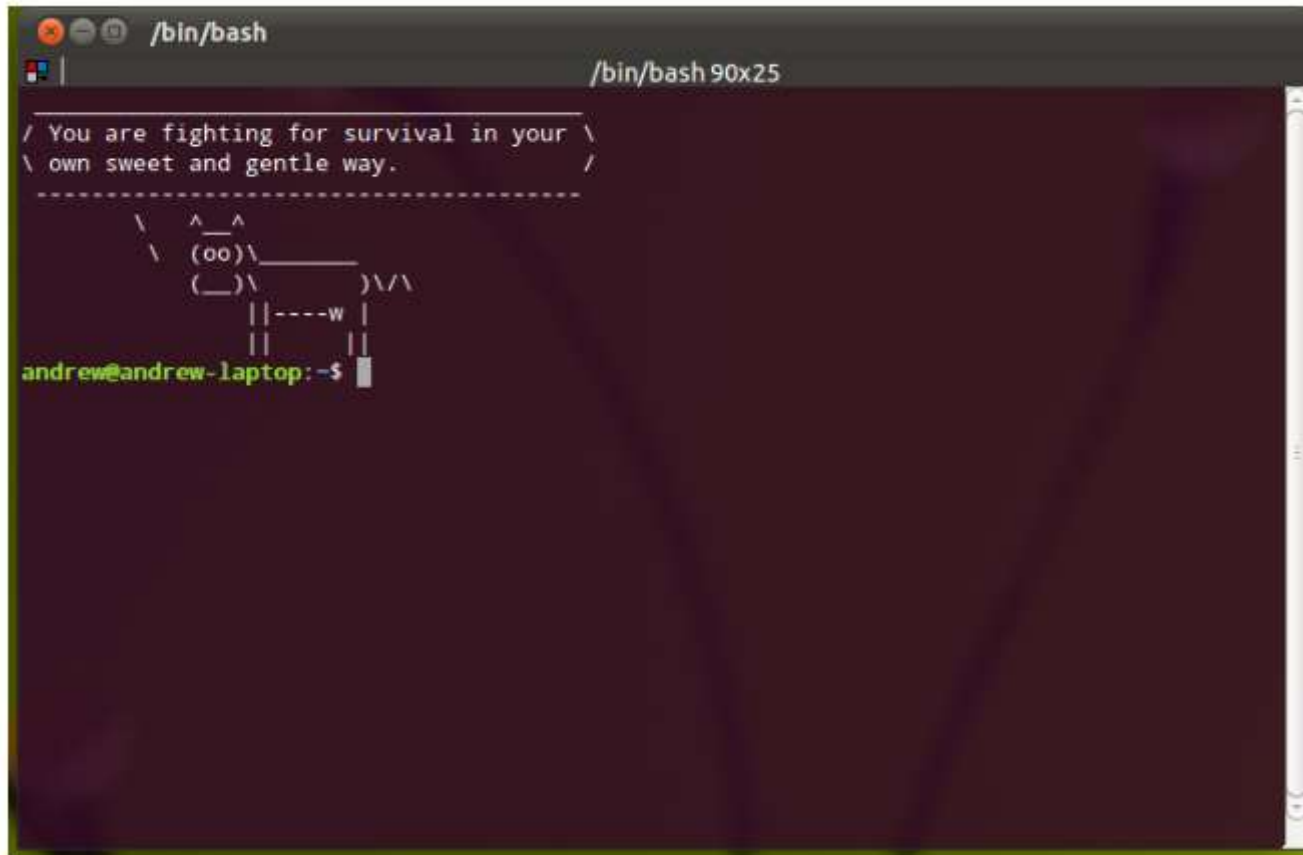
The terminal is a powerful tool for users who are comfortable with text commands. It allows for more precise control and scripting capabilities compared to graphical interfaces. In Windows, the equivalent would be the Command Prompt or PowerShell. In Ubuntu it is called a Bash terminal.



What is Bash

Bash, short for "Bourne Again SHell," is a command processor or shell that provides a command-line interface for interacting with the operating system. It's the default shell for many Linux distributions.

Bash is a powerful scripting language and a command interpreter. It allows users to automate tasks, write scripts, and execute complex commands. Bash is an integral part of the Linux ecosystem, providing a consistent interface for users and scripting.



```
/bin/bash
/ You are fighting for survival in your \
\ own sweet and gentle way. /
-----
\   ^__^
\  (oo)\_______
    (__)\       )\/\
        ||----w |
        ||     ||
andrew@andrew-laptop:~$
```

Why Learn Bash

Automation: Bash scripts can be written to automate repetitive tasks in data preprocessing, analysis, or simulations.

Remote Servers: Researchers often work with remote servers for high-performance computing or cloud computing. Bash commands are crucial for managing files and executing tasks on these servers.

Version Control: When using version control systems like Git, Bash commands are used to interact with repositories and manage code versions.

Data Exploration: For exploring and manipulating data, researchers might use Bash commands in combination with tools like awk and sed to process and analyze text-based data files efficiently.

In summary, Bash commands are integral for researchers to navigate, organize, and manipulate files and directories efficiently, both locally and on remote servers. These skills contribute to a more streamlined and reproducible research workflow.

What will we do ?

Up to this point we have been working with a lot files, namely, Python files and data files. For the most part we have been using the Windows explorer, or the alternative if your on Mac or Linux.

However, an important skill as a researcher these days is to understand how to navigate a terminal. In most cases it involves file management and running scripts.

All apps revolve around terminals or what we call a CLI - command line interface. Anaconda, Python, Slack, Canvas, AWS, all these services give you terminal or CLI access so you have more control over a visual interface like apps. It what makes the digital world go round, so to speak

GitBash

In windows, we can easily edit, copy, rename, move and delete these files. We can also create files and folders. All with a click of a mouse!












Terminals do not have a mouse, so what do you do.

You need to type out the instructions!!! Oh no more typing :(... yes more typing. But just like we tried to explain with writing Python scripts, we are recording the steps taken to read in the data, filter, sort and plot it. In excel, you don't track all these steps. Therefore it is difficult to rememeber what you did 6 months later if you suddenly want to change something.

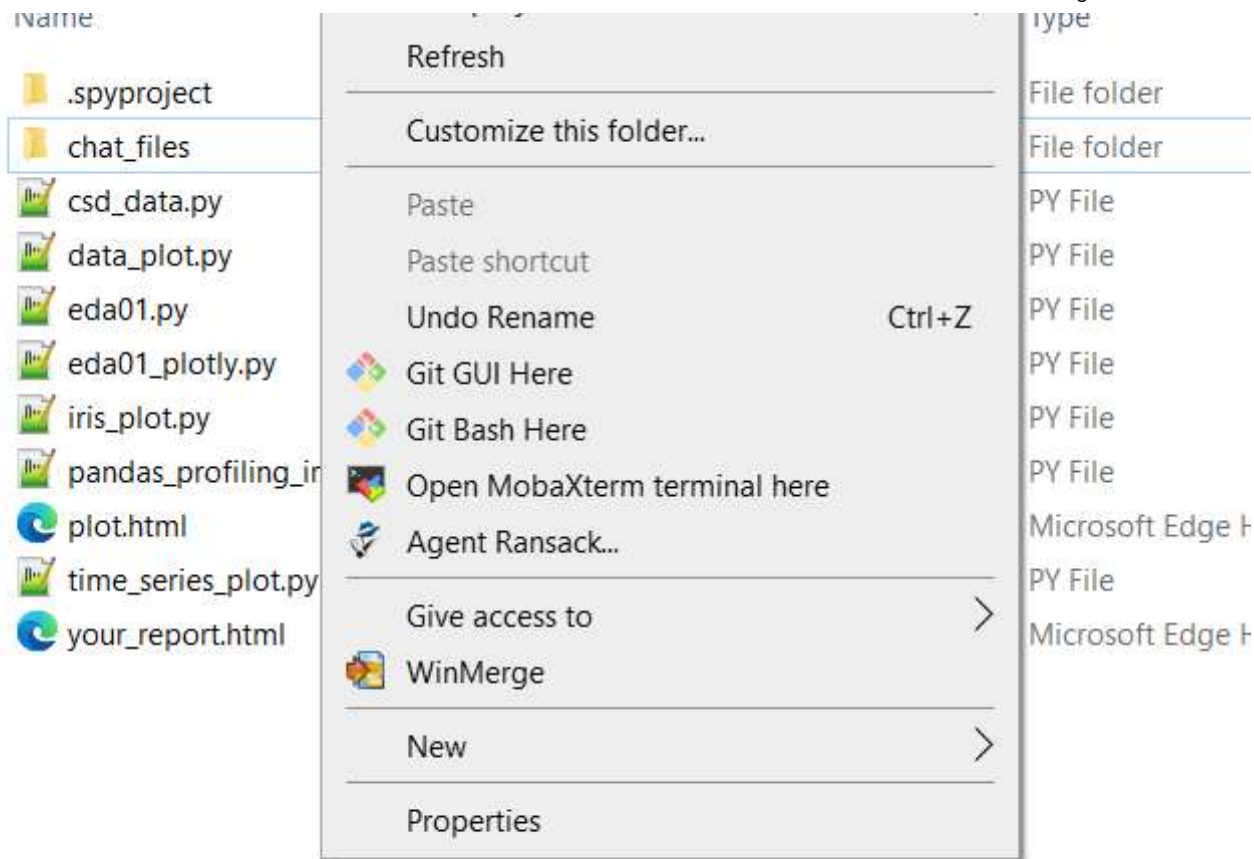
Now let us assume we have 100s of files, and you want to rename them? Are you going to click on each file in windows and rename it...you might of had to do this or something similar. Just like in Python where we try to automate the boring and mundane tasks, using a Bash terminal we can automate file management.

There are many things we can do with a bash terminal, however we will just focus on file management for now.

Let us go to yesterdays project we created with Spyder:

Name	Date modified	Type	Size
 .spyproject	31-Jan-24 10:06 AM	File folder	
 chat_files	31-Jan-24 12:09 PM	File folder	
 csd_data.py	31-Jan-24 12:16 PM	PY File	1 KB
 data_plot.py	31-Jan-24 12:06 PM	PY File	1 KB
 eda01.py	31-Jan-24 10:36 AM	PY File	1 KB
 eda01_plotly.py	31-Jan-24 10:39 AM	PY File	1 KB
 iris_plot.py	31-Jan-24 11:31 AM	PY File	1 KB
 pandas_profiling_iris.py	31-Jan-24 12:21 PM	PY File	1 KB
 plot.html	31-Jan-24 10:39 AM	Microsoft Edge HT...	3,605 KB
 time_series_plot.py	31-Jan-24 11:42 AM	PY File	1 KB
 your_report.html	31-Jan-24 5:24 PM	Microsoft Edge HT...	1,056 KB

Now in this same folder, right-click on an empty space and click on "Git-bash" as one of the options:

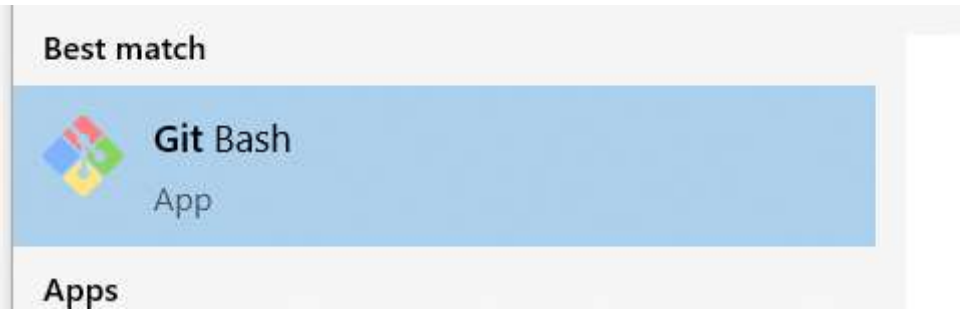


The Git-bash terminal will now open:



```
MINGW64: c:/Users/BBarsch.CSIR/css2024_day03
(base)
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03
$
```

You can also go to Windows app and search for it too:



Either way the Bash terminal will pop up and if you selected the it from your Spyder project folder it should show the default text that appears every time, this is known as your terminal prompt where you will input commands and get outputs:

```
(base)
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03
$
```

It is not necessary to know what this all means but in summary the prompt is telling you that you are in the base conda environment, logged in a user "CSIR+BBarsch" on machine "BBARSCH-NB1" using the MINGW64 shell, and your current working directory is "css2024_day03." The shell is ready to accept commands (indicated by the \$ symbol).

> Your prompt will not look the same as mine, since you will have a different user, machine name, and project folder. That is fine!

The main part of the prompt to take note of is this part: `~/css2024_day03` - This again represents the current working directory. In this example, it's the "css2024_day03" directory located in the user's home directory (~ represents the home directory).

> If you opened the Bash terminal via windows apps you won't see your project folder, you will see the following without any folder reference:

...

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~
$
```


That is fine, we will figure out how to move around to different folders.

Basics Terminal Functions

Just like in Python we had interactive console which we used to run a few lines of code, add numbers and print some text to the screen. Well we can do the same in a Bash terminal too. Instead of the `print()` function in Python we use `echo`. Input the following:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03
$ echo "hello world"
hello world
```

To do basic arithmetic you will also use the `echo` command like this:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03
$ echo $((5+5))
10
```

When you enclose an arithmetic expression within `$(())`, the shell evaluates the expression and substitutes the result, otherwise it would treat as a string rather. There are other ways to do arithmetic.

You can also store variables and create a simple equation:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03
$ x=5
(base)
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03
$ y=10
(base)
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03
```

```
$ sum=$((x+y))  
(base)  
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03  
$ echo "The sum is: $sum"  
The sum is: 15
```

Note that `x = 10` will give you an error, bash is sensitive to spaces

Directories

Getting info about the current working directory others around it:

1. pwd (print current working directory)

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03  
$ pwd  
/c/Users/BBarsch.CSIR/css2024_day03
```

Type this command always to check where you are...so you know where you are going

Explore directories

Input the following in the terminal to list files and folders:

1. ls (list contents of current directory)

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03  
$ ls  
chat_files/      iris_plot.py  
csd_data.py      pandas_profiling_iris.py  
data_plot.py     plot.html
```

```
eda01.py          time_series_plot.py
eda01_plotly.py   your_report.html
```

2. ls -lh (list contents of current directory, -lh are extra parameters to give details for each file and make it human readable)

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03
$ ls -lh
total 4.6M
drwxr-xr-x 1 CSIR+BBarsch 4096  0 Jan 31 12:09 chat_files/
-rw-r--r-- 1 CSIR+BBarsch 4096 410 Jan 31 12:16 csd_data.py
-rw-r--r-- 1 CSIR+BBarsch 4096 608 Jan 31 12:06 data_plot.py
-rw-r--r-- 1 CSIR+BBarsch 4096 799 Jan 31 10:36 eda01.py
-rw-r--r-- 1 CSIR+BBarsch 4096 601 Jan 31 10:39 eda01_plotly.py
-rw-r--r-- 1 CSIR+BBarsch 4096 694 Jan 31 11:31 iris_plot.py
-rw-r--r-- 1 CSIR+BBarsch 4096 333 Jan 31 12:21 pandas_profiling_iris.py
-rw-r--r-- 1 CSIR+BBarsch 4096 3.6M Jan 31 10:39 plot.html
-rw-r--r-- 1 CSIR+BBarsch 4096 537 Jan 31 11:42 time_series_plot.py
-rw-r--r-- 1 CSIR+BBarsch 4096 1.1M Jan 31 17:24 your_report.html
```

3. ls -lR (list contents of current directory, -lR are extra parameters to give details for each file sub folders)

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03
$ ls -lR
.:
total 4681
drwxr-xr-x 1 CSIR+BBarsch 4096  0 Jan 31 12:09 chat_files/
-rw-r--r-- 1 CSIR+BBarsch 4096 410 Jan 31 12:16 csd_data.py
-rw-r--r-- 1 CSIR+BBarsch 4096 608 Jan 31 12:06 data_plot.py
-rw-r--r-- 1 CSIR+BBarsch 4096 799 Jan 31 10:36 eda01.py
-rw-r--r-- 1 CSIR+BBarsch 4096 601 Jan 31 10:39 eda01_plotly.py
```

```
-rw-r--r-- 1 CSIR+BBarsch 4096      694 Jan 31 11:31 iris_plot.py
-rw-r--r-- 1 CSIR+BBarsch 4096      333 Jan 31 12:21 pandas_profiling_iris.py
-rw-r--r-- 1 CSIR+BBarsch 4096 3691416 Jan 31 10:39 plot.html
-rw-r--r-- 1 CSIR+BBarsch 4096      537 Jan 31 11:42 time_series_plot.py
-rw-r--r-- 1 CSIR+BBarsch 4096 1080957 Jan 31 17:24 your_report.html

./chat_files:
total 4508
-rw-r--r-- 1 CSIR+BBarsch 4096 292998 Jan 31 11:48 'Octupole derfomation nuclei Data (2).x
lsx'
-rw-r--r-- 1 CSIR+BBarsch 4096 34641 Jan 31 12:09  csd_complexes_geometric_properties_for
_activation_energies_for_css_24.xlsx
-rw-r--r-- 1 CSIR+BBarsch 4096 4282979 Jan 31 11:47  poker_2016_09_27.csv
```

4. ls chat_files/

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03
$ ls -l chat_files/
total 4508
-rw-r--r-- 1 CSIR+BBarsch 4096 292998 Jan 31 11:48 'Octupole derfomation nuclei Data (2).x
lsx'
-rw-r--r-- 1 CSIR+BBarsch 4096 34641 Jan 31 12:09  csd_complexes_geometric_properties_for
_activation_energies_for_css_24.xlsx
```

5. ls ..

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03
$ ls ..
'3D Objects'/'
```

```
'Start Menu'@  
Anaconda3/  
Templates@  
AppData/  
Untitled.ipynb
```

6. ls ../../

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03  
$ ls ../../  
Administrator/  'All Users'@  BBarsch/
```

7. ls ../css2024_day02/

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03  
$ ls ../css2024_day02/  
chat_files/  data_02/  day2.xlsx  day2_file.py  day2_file2.py  output/  plot.html  plots.py  
pulsar.csv  your_report.html
```

Anytime you see a backslash at the end of a text it refers to a folder: `css2024_day02/`

Note, when we are exploring other directories with `ls` we are using relative paths compared to our current working directory, just like we did in Python with `read_csv`. We can also use absolute paths like:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02/temp  
$ ls /c/Users/BBarsch.CSIR/css2024_day03  
chat_files/  data_plot.py  eda01_plotly.py  pandas_profiling_iris.py  time_series_plot.py  
csd_data.py  eda01.py      iris_plot.py     plot.html                your_report.html
```

Moving Around

Let us move to the directory: `css2024_day02/`.

First where are we, check with `pwd`:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day03
$ pwd
/c/Users/BBarsch.CSIR/css2024_day03
```

Where do we need to go: `../css2024_day02/` this means it is one folder up.

So we use the command `cd` or change directory to move to that directory:

```
$ cd ../css2024_day02/
```

Then you will see the prompt shows `css2024_day02` now:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02
$
```

And to double check use `pwd`:

```
$ pwd
/c/Users/BBarsch.CSIR/css2024_day02
```

Check the contents of the folder:

```
$ ls
chat_files/  data_02/  day2.xlsx  day2_file.py  day2_file2.py  output/  plot.html  plots.py
```

pulsar.csv your_report.html

Creating folders & files

Let us create a new folder called **temp** with **mkdir** and list the contents of the directory:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02
$ mkdir temp
(base)
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02
$ ls
chat_files/  data_02/  day2.xlsx  day2_file.py  day2_file2.py  output/  plot.html  plots.py
pulsar.csv  temp/    your_report.html
(base)
```

Let us change directories to **temp**:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02
$ cd temp
(base)
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02/temp
```

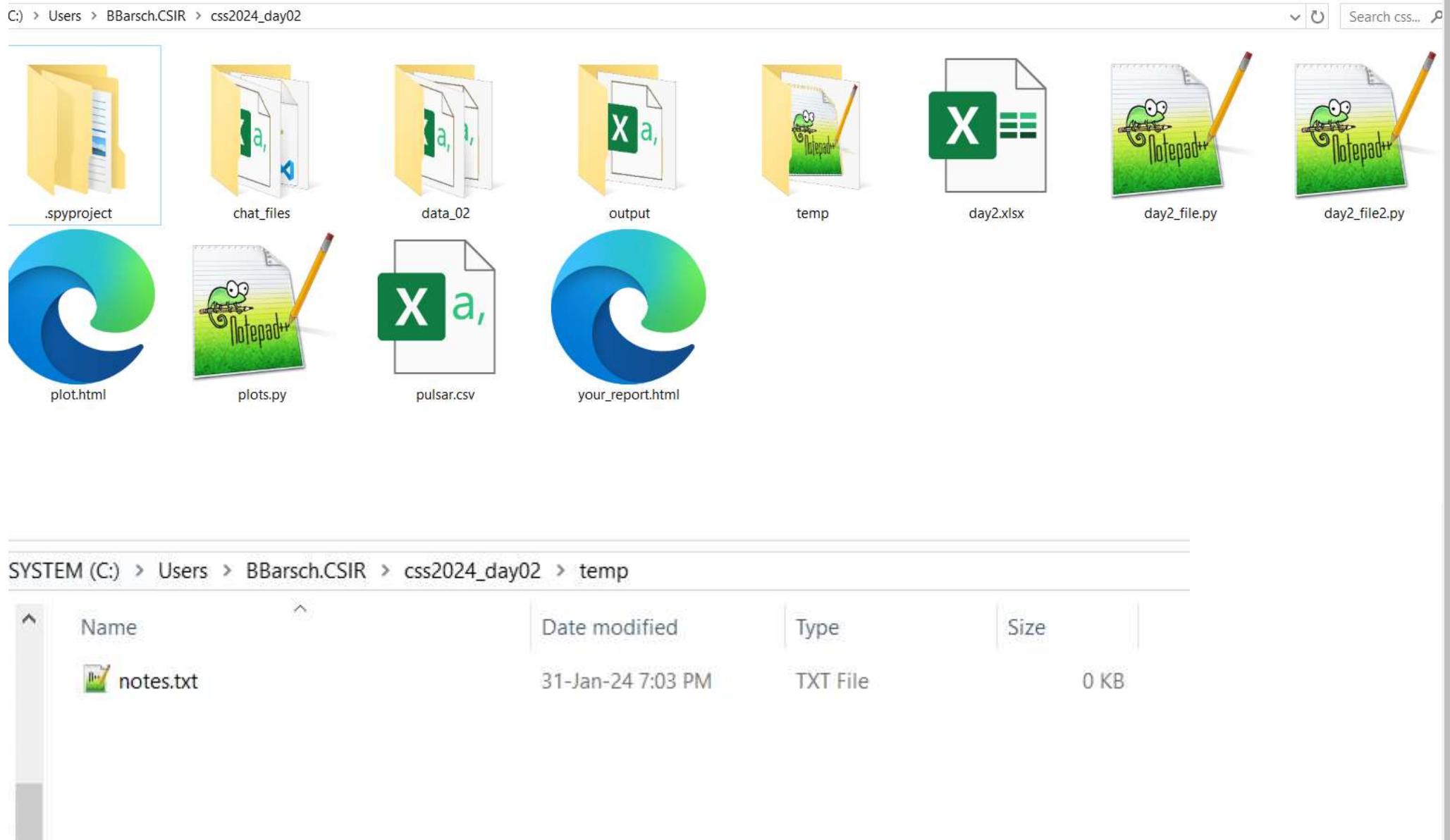
Let us create a file called **notes.txt** with **touch**:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02/temp
$ touch notes.txt
(base)
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02/temp
```



```
$ ls  
notes.txt
```

What does this look now in windows explorer:



Copy, rename, move, & delete files

Let us make another copy of `notes.txt` with `cp` command:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02/temp
$ cp notes.txt notes2.txt
(base)
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02/temp
$ ls
notes.txt  notes2.txt
```

Let us rename `notes.txt` to `notes1.txt` with the `mv` command:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02/temp
$ mv notes.txt notes1.txt
(base)
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02/temp
$ ls
notes1.txt  notes2.txt
```

The besides renaming files with `mv` you can also move it files. Let us move `notes2.txt` one folder up:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02/temp
$ mv notes2.txt ../
(base)
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02/temp
$ ls
notes1.txt
(base)
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02/temp
```

```
$ ls ../  
chat_files/  data_02/  day2.xlsx  day2_file.py  day2_file2.py  notes2.txt  output/  plot.ht  
ml  plots.py  pulsar.csv  temp/  your_report.html  
(base)
```

We can delete a file with the `rm` command, let us delete notes1.txt:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02/temp  
$ rm notes1.txt  
(base)  
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02/temp  
$ ls
```

We can do the same operations for copy `cp` and remove for `rm` folders, however in each case they need an extra parameter added to indicate this is a recursive operation. As it won't just copy the folder but each of the files inside it, and in the same removing files inside folders.

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02  
$ cp -r temp temp2  
(base)  
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02  
$ ls  
chat_files/  day2.xlsx      day2_file2.py  output/      plots.py      temp/      your_report.html  
data_02/     day2_file.py  notes2.txt     plot.html    pulsar.csv    temp2/
```

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02  
$ rm -r temp2/
```

Moving Multiple Files

Now what if you want to move multiple files. Let us say we want to move all the python files to a new folder called `py_folder`. We first need create that folder and move it using the following:

```
mkdir py_folder
mv *.py pyfolder
```

```
$ ls -l py_folder/
total 8
-rw-r--r-- 1 CSIR+BBarsch 4096  0 Jan 30 09:49 day2_file.py
-rw-r--r-- 1 CSIR+BBarsch 4096 3987 Jan 30 15:40 day2_file2.py
-rw-r--r-- 1 CSIR+BBarsch 4096  813 Jan 30 21:52 plots.py
```

How did we do that, we did it with the `*` wildcard. It is used to match zero or more characters in a file name or a path. It allows us to search for a pattern for any filename that just ends with `.py`. You can also specify certain characters as well if you are searching for certain filenames. For example, `?` wildcard matches any single character. For example, `ls a?b.txt` would match any file with a name that has a “b” surrounded by an “a” and any single character, such as “a1b.txt” or “aXb.txt”.

Interacting with Files

Assuming we are still in "css2024_day02/" folder, we can view any of the contents of the files in the terminal too by using the `cat` command. Let us view the `pulsar.csv` file:

```
CSIR+BBarsch@BBARSCH-NB1 MINGW64 ~/css2024_day02
$ cat pulsar.csv
,sepal_length,sepal_width,petal_length,petal_width,class,sepal_length_sq,sepal_length_sq_2
100,6.3,3.3,6.0,2.5, virginica,39.69,39.69
101,5.8,2.7,5.1,1.9, virginica,33.64,33.64
102,7.1,3.0,5.9,2.1, virginica,50.41,50.41
103,6.3,2.9,5.6,1.8, virginica,39.69,39.69
104,6.5,3.0,5.8,2.2, virginica,42.25,42.25
```

```
105,7.6,3.0,6.6,2.1, virginica,57.76,57.76  
107,7.3,2.9,6.3,1.8, virginica,53.29,53.29  
108,6.7,2.5,5.8,1.8, virginica,44.89,44.89  
109,7.2,3.6,6.1,2.5, virginica,51.84,51.84  
110,6.5,3.2,5.1,2.0, virginica,42.25,42.25
```

If it is very large file with 1000s of lines you can use `head` to see the first 10 lines, and `tail` to see the last 10 lines of a file.

We can also redirect content to a file using the append ">>" operator. This is useful for logging information:

```
echo "2024-02-01,80" >> logfile.txt
```

The file will be created if it did not exist. If there are cases where we want to redirect outputs to a new file each time we can use the ">" redirect operator.

Now what happens if you want to edit a file itself. We can use a built in terminal editor called Nano to do that. Input the following:

```
nano notes2.txt
```

GNU nano 6.4

notes2.txt

[Read 0 lines]

^G Help	^O Write Out	^W Where Is	^K Cut	^T Execute	^C Location	M-U Undo	M-A Set Mark	M-] To Bracket
^X Exit	^R Read File	^_ Replace	^U Paste	^J Justify	^/ Go To Line	M-E Redo	M-6 Copy	^Q Where Was

Just like any word or notepad text editor we can add the following text:

```
hello world
Linux was invented in 1991 by Linus Torvalds
Linux is written mostly in the C programming language
```

MINGW64:/c/Users/BBarsch.CSIR/css2024_day02

GNU nano 6.4

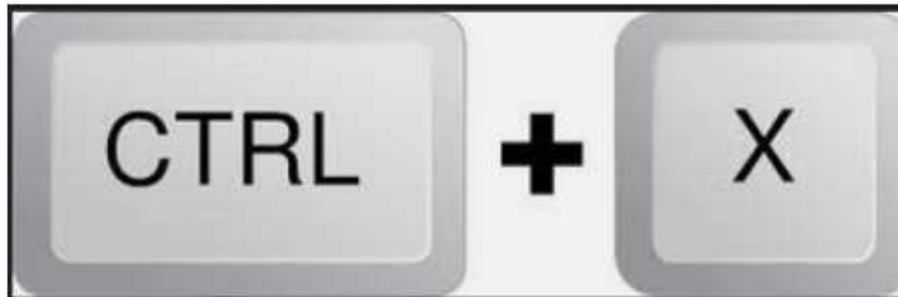
notes2.txt

hello world

Linux was invented in 1991 by Linus Torvalds

Linux is written mostly in the C programming language

We also have several options at the bottom of the screen to Get Help, Exit, and many more. The ^ refers to the Ctrl key on your keyboard. To save the changes we made to the file we need to press Ctrl key and the x key on your keyboard:



When you do that the bottom of the screen will change to this:

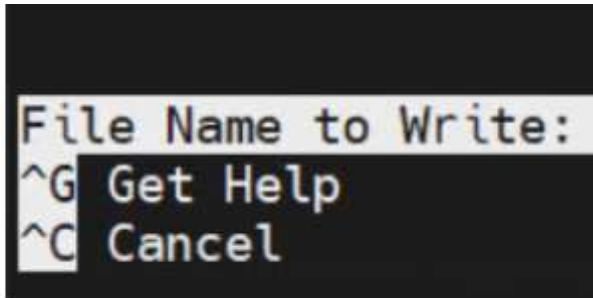
Save modified buffer? (Answering "No" will DISCARD changes.)

Y Yes

N No

^C Cancel

To save changes press the y key or n to discard the changes. Press the y key to save changes and you will see the following:



Now just press the Enter key and you will exit the text editor and back to your terminal.

Find & Search

There are cases where you may want to find files themselves. For example all files that start with day in it:

```
$ find -name "day*"
./day2.xlsx
./py_folder/day2_file.py
./py_folder/day2_file2.py
```

It found one excel file in the current directory and two python files in a sub-directory.

Now there may be cases where you want to search for specific words in files, and you can use "grep" for that. For example if we want to find all rows that have the word "virginica" in it we can do it as follows:

```
$ grep -n "virginica" *.csv
101:6.3,3.3,6.0,2.5,virginica
102:5.8,2.7,5.1,1.9,virginica
103:7.1,3.0,5.9,2.1,virginica
```

Again using the * wildcard to search through all csv files.

Summary

There are many different uses for a bash terminal, some might not be relevant to you know but it is an essential skill. In summary, Bash commands are integral for researchers to navigate, organize, and manipulate files and directories efficiently, both locally and on remote servers. These skills contribute to a more streamlined and reproducible research workflow.

There is still a lot of content we did not cover. Like how to connect to remote servers and copy from them. Writing bash script files to automate processes and much more! However, this is sufficient as introduction to bash terminals.

Data & File Management

Review these three videos:

Knowledge clip: Keeping research data organized - <https://www.youtube.com/watch?v=YsIfY4W-NAg>  <https://www.youtube.com/watch?v=YsIfY4W-NAg>



(h
tt
p:
//s
a
v
ef
ro
m
.n
et
/?
ur
l=
ht
tp
s
%
3
A
%
2
F
%
2
F
w
w
w.

y
o
ut
u
b
e.
c
o
m
%
2
F
w
at
c
h
%
3
F
v
%
3
D
Y
sl
fY
4
W
-
N

A
g
&
ut
m
—
s
o
ur
c
e
=
c
h
a
m
el
e
o
n
&
ut
m
—
m
e
di
u
m
=

e
xt
e
n
si
o
n
s
&
ut
m
—
c
a
m
p
ai
g
n
=I
in
k
—
m
o
di
fi
er

[v=YsIfY4W-NAg\)_](#))

Knowledge clip: Data Management Plans (DMPs) - <https://www.youtube.com/watch?v=GRNsLTQGjCo>  <https://www.youtube.com/watch?v=GRNsLTQGjCo>




(h
tt
p:
//s
a
v
ef
ro
m
.n
et
/?
ur
l=
ht
tp
s
%
3
A
%
2
F
%
2
F
w
w
w.

y
o
ut
u
b
e.
c
o
m
%
2
F
w
at
c
h
%
3
F
v
%
3
D
G
R
N
s
L
T
Q

Gj
C
o
&
ut
m
—
s
o
ur
c
e
=
c
h
a
m
el
e
o
n
&
ut
m
—
m
e
di
u
m

=
e
xt
e
n
si
o
n
s
&
ut
m
—
c
a
m
p
ai
g
n
=|
in
k
—
m
o
di
fi
er

[v=GRNsLTQGjCo\)_](#))

Knowledge clip: What is Research Data Management (RDM)? - <https://www.youtube.com/watch?v=bbsLmy3Njv4> 



(h
tt
p:
//s
a
v
ef
ro
m
.n
et
/?
ur
l=
ht
tp
s
%
3
A
%
2
F
%
2
F
w
w
w.

y
o
ut
u
b
e.
c
o
m
%
2
F
w
at
c
h
%
3
F
v
%
3
D
b
b
s
L
m
y
3

Nj
v
4
&
ut
m
—
s
o
ur
c
e
=
c
h
a
m
el
e
o
n
&
ut
m
—
m
e
di
u
m

=
e
xt
e
n
si
o
n
s
&
ut
m
—
c
a
m
p
ai
g
n
=|
in
k
—
m
o
di
fi
er

(<https://www.youtube.com/watch?v=bbsLmy3Njv4>)_)

