# How to run Python scripts multiple times

## Summary:

- I will provide a tutorial in creating and running a Bash script to be able to run a Python script multiple times using a **for loop**.

## Prerequisites:

- Familiarity with Python creating scripts in editors and running scripts in the command line
- Familiarity with command line arguments

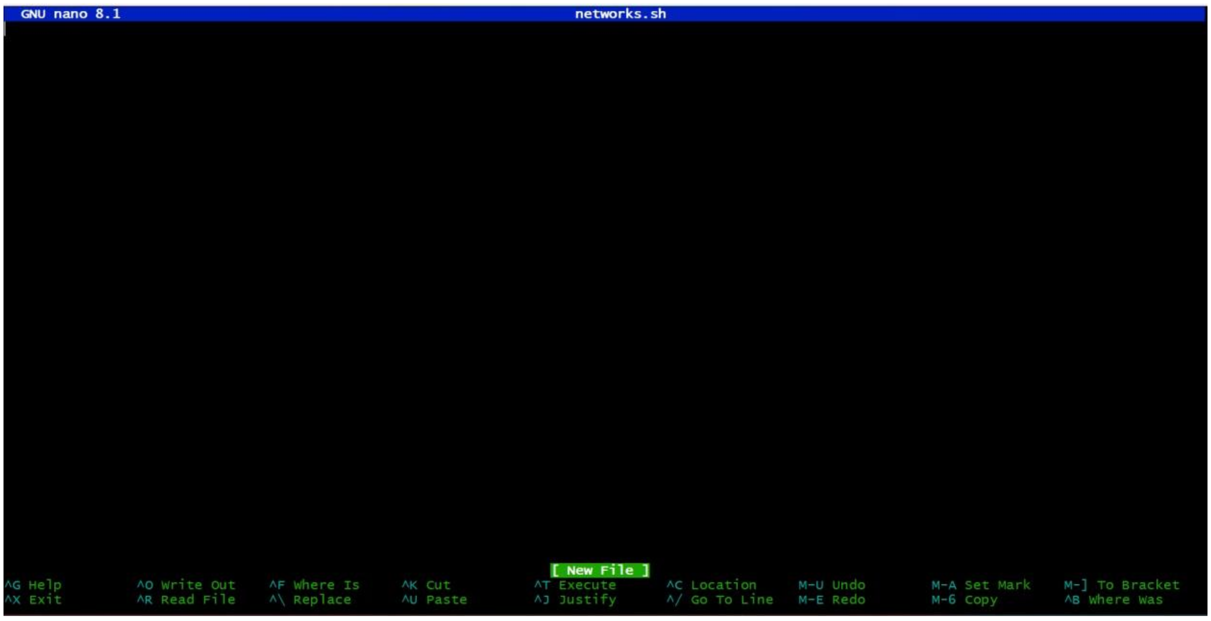
## Process:

### Download the packages and set up the environment

Bash script can run on Windows, Linux and MacOS. I will run this tutorial on Windows. On Windows, the app needed is Git Bash. You can download Git Bash from here: https://git-scm.com/downloads
After downloading it, the environment needs to be set up, by navigating to the directory containing all the necessary files is located. To go to the directory, the **cd ‘name of directory’** command is used.
In the directory, I use **ls** to see the list of the files.

### Creating a Bash script

The script can be created in several ways. One way to do to is by typing **nano filename.sh** and pressing **Enter** into the command line of Git Bash environment, which creates and opens a new script file – fig. 1.



```
GNU nano 8.1                                   networks.sh

                                              [ New File ]
```

*Fig. 1. The empty new file*

## For loops in Bash

The next sections explore how, after creating and opening a file, to use Bash syntax, such as variables and for loops, to write the script. At the beginning of script, the **!#/bin/bash** line of code indicates that to run the script Bash needs to be used. A for loop runs is running code repeatedly until a specific condition has been satisfied. For example, maybe you want to print all the elements from a sequence of elements, one by one (the condition to be satisfied is that the loop runs as long as there are elements in the sequence).

```
#!/bin/bash
elements=('1', '2', '3' ,'4')
for element in ${elements[@]}
do
    echo "$element"
done
```

Before the loop, a sequence of elements called **elements** was initialized as a variable.
If there is a space between the name of the variable and its value, Bash throws an error.
On the next line, the for loop is started. The first line indicates the current element of the sequence at which the process came. In the **${elements[@]}**, **@** indicates the index of the value of the element in the sequence, while the **$** is how you call a variable.

## Variables and command-line arguments

I have started to explore the variables already and one example was the sequence of elements. However, if you have a Python script in your Bash script, you can have Python variables with values that are taken from the Bash script. For this, make sure that your Python variables are also command-line arguments in the Python script by using the **argparse** module (more details on how to use the module at this link: argparse — Parser for command-line options, arguments and sub-commands — Python 3.12.5 documentation). When running it in Bash, you specify that the value of a Python command-line argument is a value from the sequence of elements mentioned above.

## Example of a Bash script

My Bash script contains a Python script that does historical network analysis by exploring a dataset of Quaker missionaries in the 17th century (more details about historical network analysis in Python and the dataset I used are found at this link: https://programminghistorian.org/en/lessons/exploring-and-analyzing-network-data-with-python . I use Bash to find the occurrences in the datasets meeting the criteria of being male and being born in a 17th century decade. The script has similar elements of Bash syntax as those covered above, such as a for loop, the decades initialized as a sequence variable named **years**. Within the for loop, however, aside from printing the decade (as a variable **year**) and a dashed line to separate the iterations, the **year** also serves as a value of the Python script command line argument **birth_year** alongside the argument of **gender**, with the value **male.**

```
#!/bin/bash
#loop to run the python script multiple times
years=('1610' '1620' '1630' '1640' '1650' '1660' '1670' '1680' '1690' '1700')
for year in ${years[@]}
do
    py network_python.py --birth_year $year --gender male
    echo "Year per iteration:$year"
    echo "---------------------"
done
```

## Running the script

Now, I have to run my script. A method to do this is to type in **bash filename.sh** and press **Enter.** I wanted to print the names and birth years of the people born in the decades mentioned in the Bash script. An example is **John Swinton** in the second iteration. There is also a warning from the Python script but which does not affect the output.

## Take-away points:

- To code in Bash, you need to download an app called Git Bash
- Git Bash runs on Windows
- You can use **ls** in order to list the files in a directory in the command line
- You can use **cd 'directory name'** in order to change directory
- When creating a Bash script file, you can use **nano filename.sh**
- Do not leave spaces between the name of the variable and its value
- You can have a Python script with command-line arguments within a for loop in Bash
- When running a Bash script file, you can use **bash filename.sh**