Fundamentals of Agro-Environmental Data Science

Exercise 1 - Manipulate files and directories from the command line

This is a learning/reading exercise, with several tasks. You should repeat all the examples given in your system, and check that you obtain equivalent results.

Conventions of this document

In all commands, `$` indicates that it should be run at the operating system terminal. However, `$` is not part of the command and should not be included in the instruction.

Introduction

Many of the examples of Data Science projects that you will find online were created in a operating system that is different from yours. Additionally, in many cases, the environment that you will have available to run your projects (either a remote virtual machine, an application in the browser or a collaborative environment as Colab) will be different from yours. In many cases, you will not have access to a graphic interface, with windows, panels and buttons, but only a terminal with a command line interface (CLI).

Because of this, you should be able to execute some basic operations using the CLI, which includes:

- navigate through directories
- list files
- create files and directories
- copy, delete and rename files and directories
- identify the current location
- identify the path to a file
- run a python script
- reuse commands
- connect to remote computers

Using the terminal with bash

Speed up your workflow

One of the reasons you should use the terminal is because it will speed up greatly your workflow in data science projects. You might find strange this statement, considering that in the terminal you need to write your commands by hand. And some of these seem endless...

However, two practices speed up the use of the terminal, and sometimes even make it easier to use:

- pressing the ***TAB key*** after writing a few letters of a command or a file will allow the system to autocomplete or show options of what you are trying to write

- you can use the ***history*** to reuse a command written before. You just need to press the ***upper arrow*** in your keyboard until you find what you need.

Another reason to use the terminal, particularly in data science projects, is that you have an higher power of parameterization in running scripts. Many scripts can only be run from the command line, because no graphic interfaces are created. Also programmers make a lot of customizations available through parameters in the command line, but they do not translate these into graphic interfaces. Even if you're using a graphic interface as Jupyter Notebook or R-Studio to run your data science scripts, command are essentially run as it would in the command line.

It is, therefore, an important skill being able to manage files and run scripts from the terminal.

Bash as the language in the terminal

Linux and macOS systems have ***bash*** as the terminal language by default. Windows systems do not. Windows uses, instead, the ***Command shell*** (CMD), and ***PowerShell***. Although any of these can achieve what we want from this exercise, we will focus on ***bash*** because of its wider use in the data science community.

If your system is linux or macOS, you most probably do not need to do anything to have it ready. Just open a terminal and run

```
$ ls
```

The result should be a list of files in the current directory (or folder).

If you're running Windows, you have two options:

- find and run the equivalent command to bash for Command (CMD) shell
- install ***bash*** to run commands as you would do in linux or macOS.

Bash can be installed as a part of Git installation. At the installation steps of Git, make sure to select the following options:

- 5th installation screen, **Adjust your PATH environment**, leave **"Git from the command line and also from 3rd party software"**
- 8th installation screen, **Configuring the line ending conversions**, leave **"Git from the command line and also from 3rd party software"**
- 9th installation screen, **Configuring the terminal emulator to use with Git Bash**, select **"Use Windows' default console window"**

After the install completes, to open your bash terminal, search and run `Git Bash`.

Prepare for the exercise

Before starting the exercise, you need to create a new file called *example.txt* in your home directory. This file can be created with Notepad in Windows, or with the following command in a bash terminal:

```
$ touch example.txt
```

Most useful bash (and CMD) commands to learn

We will show the use of the most useful commands. Remember that all commands in bash are **case sensitive**.

Print current **w**orking **d**irectory (`pwd`)

To show our current directory, in which we are positioned now, run `pwd`

```
$ pwd
/home/rfigueira
```

or in CMD, `cd` for **c**hange **d**irectory, without any parameters

```
$ cd
C:\Users\rfigueira
```

Notice the difference of the direction of the slashes in the path, forward in the case of bash (`/`) and backwards in the case of CMD (`\`).

List files and directories (`ls`)

You can make a list of files and directories in the current working directory:

```
$ ls
Desktop     Downloads   Movies      Pictures    example.txt
Documents   Library     Music       Public
```

or in CMD, `dir`

```
$ dir
```

In bash, we can do a `list` command with different options. For example, to create a vertical list with attributes, we can do

```
$ ls -l
total 8
drwx------+  3 rfigueira  staff    96 Sep  7 14:42 Desktop
drwx------@  7 rfigueira  staff   224 Sep 24 17:01 Documents
```

```
drwx------+  7 rfigueira  staff   224 Sep 24 22:52 Downloads
drwx------@ 76 rfigueira  staff  2432 Sep 23 09:03 Library
drwx------   3 rfigueira  staff    96 Sep  7 14:42 Movies
drwx------+  4 rfigueira  staff   128 Sep 18 14:48 Music
drwx------+  4 rfigueira  staff   128 Sep 17 18:03 Pictures
drwxr-xr-x+  4 rfigueira  staff   128 Sep  7 14:42 Public
-rw-r--r--   1 rfigueira  staff    20 Sep 26 21:37 example.txt
```

In this case, the output contains more information, including if the item is a directory or a file (the **d** flag), the permissions, the owner and the group the file belongs to, the size in bytes, the last creation or editing date and the name of file.

You can list all files that have a certain pattern in the name. For example, all files with the extension *.txt*

```
$ ls *.txt
example.txt
```

The asterisk works as a wildcard, indicating any sequence of characters.

Also, try the following commands with three different options:

```
$ ls -lh
$ ls -lh Documents
$ ls -lisa
```

The first uses the parameter h (for human) to indicate the size of files in a human-friendly format. The second will show all the files inside Documents directory, which is a subdirectory of the current directory (adjust the name of the directory to your particular case). The last command will list all files, including the ones hidden, which filenames start with a point (.)

Change the current Working **D**irectory (`cd`)

`cd` is equal in bash and in CMD. To change to the directory Documents, which is a subdirectory of the current directory, we can do the following (try to press TAB after entering the first characters of Documents):

```
$ cd Documents
```

There is no output, except that the prompt of the terminal will indicate the directory where we are, changed. Confirm that the path changes, with `pwd` for bash or `cd` for CMD.

```
$ pwd
/Users/rfigueira/Documents
```

If we want to go one level up in the directory tree, we can use `cd ..`, where the two points indicate one level up:

```
$ cd ..

$ pwd
/Users/rfigueira
```

In the command `cd`, if we use only one point, it means the current directory:

```
$ pwd
/Users/rfigueira

$ cd .

$ pwd
/Users/rfigueira
```

And if we do `cd ../..`, or `cd ..\..` (in CMD, with backward slashes), then the cursor will place us two levels up

```
$ pwd
/Users/rfigueira/Documents

$ cd ../..

$ pwd
/Users
```

Create a new directory (`mkdir`)

Change to your home directory

```
$ pwd
/Users

$ cd ~

$ pwd
/Users/rfigueira
```

There is no direct correspondent command in CMD. You have to navigate with one or several cd commands.

Now, let's create a new directory called *project-1*

```
$ mkdir project-1
```

The command in CMD is equal. We can check that we have a new directory:

```
$ ls
Desktop     Downloads   Movies      Pictures    example.txt
Documents   Library     Music       Public      project-1
```

Full and relative paths

It is important to note that a filename, for the machine, is always composed by its full path - sequence of directories from the root level of your drive up to the current location of the file. Another important notion is that directories are also files that have a flag `d` indicating it is a directory.

Therefore, we can refer to a file using its full path, for example:

```
$ pwd
/Users/rfigueira

$ ls /Users/rfigueira/example.txt
/Users/rfigueira/example.txt
```

However, we can also refer to a file using its relative path. The relative path is the path from our current directory to the place where the file is. For example, to list a file in the current directory, the two commands are equivalent:

```
$ ls example.txt
example.txt

$ ls ./example.txt
./example.txt
```

In fact, in the first command, the `./` part is omitted because it is there by default.

Now, let's try to list a file that is not in the current working directory. First. we need to change to the directory *project-1*, and, after that, try to list the file *example.txt*

```
$ cd project-1

$ ls example.txt
ls: example.txt: No such file or directory
```

The error is expected. We can do a list command, but indicating the relative path. We will use the notion of points (`..`) to indicate that the file is located one level up in the directory tree:

```
$ pwd
/Users/rfigueira/project-1

$ ls ../example.txt
../example.txt
```

The use of the TAB key to autocomplete the name of the file is a good way of guessing if the file is in the expected place.

Copy a file (`cp`)

We can copy documents using `cp`:

```
$ cd ..

$ cp example.txt document.txt

$ ls
Desktop     Library     Pictures    example.txt
Documents   Movies      Public      project-1
Downloads   Music       document.txt
```

In CMD, use `copy`:

```
$ cd ..

$ copy example.txt document.txt
```

Or, we can copy and place the new document named *document.txt* inside the directory *project-1*:

```
$ cp example.txt ./project-1/document.txt

$ ls ./project-1
document.txt
```

In CMD:

```
$ copy example.txt .\project-1\document.txt
```

In the above commands, the point in the destination file indicates that the path starts at the current directory.

Copy a directory, including its contents (`cp -r`)

We can copy directories, including all of its contents. The flag -r in copy indicates that the command will execute recursively, i.e, copying a directory and all its contents. We can try this:

```
$ pwd
/Users/rfigueira

$ ls
Desktop     Library     Pictures    example.txt
Documents   Movies      Public      project-1
Downloads   Music       document.txt

$ cp -r project-1 project-2

$ ls project-2
document.txt
```

In CMD, we can use `xcopy`

```
$ xcopy /E .\project-1 .\project-2
```

Notice that in CMD we need to provide a relative or a full path of the directory. The optional parameter `/E` means it should copy directories and subdirectories, including empty ones.

Remove a file (`rm`)

The command to delete a file is `rm`:

```
$ ls
Desktop     Library     Pictures    example.txt
Documents   Movies      Public      project-1
Downloads   Music       document.txt    project-2

$ rm document.txt

$ ls
Desktop     Library     Pictures    project-1
Documents   Movies      Public      project-2
Downloads   Music       example.txt
```

In CMD, we use `del`:

```
$ del document.txt
```

Remove a directory (`rm -r`)

To remove a directory, we need to use the flag -r, as in `cp -r`. In the following, we will delete using a wildcard in the name of the directory.

```
$ ls
Desktop      Library      Pictures     project-1
Documents    Movies       Public       project-2
Downloads    Music        example.txt

$ rm -r project-*

$ ls
Desktop      Downloads    Movies       Pictures     example.txt
Documents    Library      Music        Public
```

It removes all the directories which name starts with *project-*, wherever it contains after that. This is very powerful, but also very dangerous.

In CMD, use (`rmdir`)

```
$ rmdir project-1
```

Create text files (`touch`)

It is very handy to create an empty file, for example, a README.md, to quickly create documentation. This can be done with the `touch`command:

```
$ touch README.md
```

In CMD, there is no equivalent. A similar option will be to execute Notepad

```
$ notepad README.md
```

This will open notepad for us, where we can save the file.

Wrap-up

In this exercise, we had a quick introduction to our terminal and commands to do several tasks:

- `pwd` (bash) od `cd` (CMD) - show full path to your current working directory.
- `ls` (bash) or `dir` (CMD) - list content - files and directories - of a directory. Can be of your current directory, or another, if you give a path and name of directory
- `cd` - change directory.
- `mkdir` - create directories.
- `cp` (bash) or `copy` (CMD) - copy files.
- `cp -r` (bash) or `xcopy` (CMD) - copy directories.
- `rm` (bash) or `del` (CMD) - delete files.
- `rm -r` (bash) or `rmdir` (CMD) - delete directories.
- `touch` - create empty files.

This concludes the current exercise.

References

Palomino, Jenny, Wasser, Leah, & Joseph, Max. (2021). earthlab/earth-analytics-intro-to-earth-data-science-textbook: Earth Analytics Updated Version of the Intro Textbook (1.5). Zenodo. https://doi.org/10.5281/zenodo.4686073