Managing your code and data

One of the critical elements of good research practice is the effective management of data and code. Adopting effective tools and techniques to keep tabs on where your data is, ensuring that data is backed up, and the production of clear documentation that describes the data transformation and analysis that has been undertaken, are all skills that researchers dealing with complex environmental datasets should adopt.

Most of us will have experienced a situation where we have been working on a project, perhaps performing some analysis on a dataset and we have somehow lost the progress that had been made, either through corruption or accidental deletion of the files. Re-doing work that has been done previously is one of the most frustrating things to have to do! So, we are going to use a range of tools that will help to minimise the chances of this occurring.

Data management has been made much easier in recent years due to cloud hosting services such as Dropbox, OneDrive, etc which you can use to store your datasets in the cloud, whilst easily accessing them on your laptop/cluster PC. We strongly encourage that all the work that you do in this module should be stored on cloud-based servers. You have 5TB of space on your OneDrive account so please use it. Under no circumstances should you depend on USB sticks and external hard drives to store your files.

One of the biggest step changes of choosing to work with MATAB, R, or any other statistical or programming language is that every step in the management and analysis of your data can be documented, evaluated, and easily reversed if need be. Compare this to Excel where you might have a spreadsheet with multiple columns and tabs containing different data processing steps. It can be difficult to keep track of what action each column/sheet is performing especially after some time away from the project.

Whilst MATLAB does have a user-friendly GUI, most of the work that you will be doing will involve the Command Window or writing/running .m files. Developing code for running your analyses can be time consuming, but also very rewarding. As with all workflows there is best-practice and coding is no different. Some of the key things to remember, when writing code for your analysis is to ensure that you:

- Comment your code liberally, especially to describe tricky parts of your workflow
- Backup your work regularly (using GitHub)
- Can identify the changes made to your code, and reverse these when needed (using GitHub)
- Use built-in functions when they are available
- Use descriptive variable names.
- Build up slowly - try simple examples to ensure it works as expected before testing on a big data set
- Write scripts for each figure you need to make.

To help us manage our code effectively, we will use tools called Git and GitHub. At its heart, GitHub is a tool to facilitate collaboration. As we are working on individual projects, we won't be using half of the functionality available within the tool, just barely scratching the surface. Ideally, you will have your own laptop or desktop PC running Windows, Linux, or Mac OS to be able to use GitHub most effectively.

1. First of all you will need to create a GitHub account, see: https://docs.aws.amazon.com/codedeploy/latest/userguide/tutorials-github-create-github-account.html
2. You can now create your own repository (where you will store your code). You should ensure that this is a private repository so that others cannot see your work. You will be able to share it with me when it comes to the portfolio submission.
3. Give your repository an informative name such as GEO8026 followed by your initials e.g. 'GEO8026_MTP'. I suggest creating multiple folders within your main repository. How you organise this is up to you, but I suggest at least one folder for each of the practical's, and one for the portfolio submission (see this example). You can do this by going to your repositories webpage, clicking on 'Add file' --> 'Create new file' --> Where it says, 'Name your file', type the name of the folder you want to create followed by a forward slash e.g. 'Practical 1/'. As you are not allowed to create empty folders you will need to give the file a name e.g. 'temp', and then click 'Commit new file'. You will see a new folder will have been created with a file called temp. You can now delete this file and populate it with files related to that folder.
4. If you do not have access to a personal computer, you will use the website to upload new files and update changes to your files manually using the 'Upload files' functionality.
5. However, if you have access to your own PC, I encourage you to use a desktop application to help you effectively manage your code. This will enable you to easily commit changes to your repository. To do this you need to download and install git. See: https://www.atlassian.com/git/tutorials/install-git
6. You can then 'clone' your repository onto your laptop/PC (see: https://docs.github.com/en/repositories/creating-and-managing-repositories/cloning-a-repository). This will create a copy of the files and folders within your repository onto your PC. I advise against cloning a repository into a folder which is sync'd to the cloud e.g. OneDrive, Dropbox, as this can cause issues. If you get into the habit of committing changes to GitHub regularly you will not have to worry about losing your code.
7. Once you have cloned into a repository you will find the files and folders that are present in the GitHub repository are also present within the specified folder on your PC.
8. When you make changes or additions to the files and folders within the repository on your PC you can sync these to GitHub using the following commands within the software 'git bash' (do not use the text including after the % - these are descriptions):

```
cd C:/_git_local/GEO8026_21_22/ % navigate to your folder where the repo is stored

git pull % update the local version of a repository in case changes have been made elsewhere

git add . % gather files within the folder + subfolder(s) for sync

git commit -am "add a descriptive message here" % saves a new commit object in the local Git repository

git push origin % sends the changed files GitHub as a new commit. Old commits can still be seen and reverted
```