Fundamentals of Agro-Environmental Data Science

Exercise 2 - Use Git and GitHub to manage and share your code, including versions

This is a learning/reading exercise, with several tasks. You should repeat and run all the examples given in your system, **adapted to your case**, 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.

The commands in this document assumes a bash terminal, but you can adapt the commands to CMD terminal.

Introduction

Developing a data science project which combines data, code and analysis means that many trial and error actions will happen. Also, creating new components or modules in code might imply changes in existing parts. It is important, however, that you do not break what is working well. This can be ensured by using versions, i.e., making copies of your project to test things, which, if work, can be integrated in the main copy. This can be done through version control.

Version control is a system to manage versions by **tracking and managing changes in your files**. A version control system maintains a record of changes to code and other content. It also allows us to revert changes to a previous point in time. Additionally, it allows to collaborate on a single project, merging contributions from different collaborators.

One of the systems that allows to achieve these goals is **Git**. **Git** uses a distributed version control model. This means that it can manage not only local versions of your code and files, but also copies in other locations, as an online repository. In our course, we will use **GitHub** as the online repository

This exercise gives a very brief introduction to **git**, mainly to ensure that you have properly setup your environment and can execute the most basic operations. However, if you want to explore more, you can read these tow online resources, that were used for inspiration:

- Version Control with Git, The Carpentries
- Git and GitHub, Earth Lab

Pre-requisites

Make sure you have:

- installed **Git** in your system, as indicated in the GreenDS Welcome-Kit
- created a personal GitHub account, as indicated in the GreenDS Welcome-Kit

Set up Git

After the installation of **Git**, we need to inform who we are. This will be associated to your code contributions, and also facilitate the use of GitHub.

You will provide two global configurations to the system with the following commands (in a bash terminal):

(Adjust the following commands to your case)

```
$ git config --global user.name "Rui Figueira"
$ git config --global user.email "ruifigueira@isa.ulisboa.pt"
```

You can check that config worked:

```
$ git config user.name
Rui Figueira

$ git config user.email
ruifigueira@isa.ulisboa.pt
```

Create a local repository

Let's imagine our first data science project. It will be a simple project, with the following four files, inside the directory `my-first-project`:

```
/my-first-project
    README.md
    temp-data.csv
    temp-average.py
    temp-doc.txt
```

Create the directory using `mkdir` and files using `touch` in your system, using a bash terminal. Then, change the working directory to *my-first-project*, using `cd`. In the terminal, we can check if this directory is a git repository, with the `git status` command:

```
$ git status
fatal: not a git repository (or any of the parent directories): .git
```

The message indicates it is not a git repository, because it lacks the hidden directory `.git`. It is inside this directory that **git** manages all that is needed to do the versioning. Therefore, we need to inform git that this directory is a repository to be managed with versioning. For that, we run `git init`

```
$ git init

$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md
        temp-average.py
        temp-data.csv
        temp-doc.txt

nothing added to commit but untracked files present (use "git add" to
track)
```

Now, the result of `git status` is different, and provides information about the repository:

- the branch `main`
- the files that are part of the repository, and their status.

In this case, no file is tracked yet. Let's see how to manage this the next step.

The steps of git

The use of git implies three important steps of version control:

- `git add` - adds changed files to version control tracking.
- `git commit` - commits the changed files to create a unique snapshot of the local repository.
- `git push` - pushes the changed files from the local copy of a repository to a server or the cloud

We can add the files in our directory to the tracking system, i.e., **stage** the files:

```
$ git add README.md
$ git add temp-data.csv
$ git add temp-average.py
$ git add temp-doc.txt
```

We could have done the `git add` of all files at once, if we did `git add .` In the case you wrongly added a file to the repository, you can undo the `git add` using `git reset <file>`, or using `git rm --cached <file>`.

Check the current status:

```
$ git status
On branch main
```

```
No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   document.csv
        new file:   temp-average.py
        new file:   temp-data.csv
        new file:   temp-doc.txt
```

The next step is to commit the files. This will create a new version of the repository. We will commit all files at once, with the wildcard `*` (or the point). When doing commit, we need to provide a message to document what we are doing, with the parameter `-m`:

```
$ git commit * -m "initial commit"
[main (root-commit) 4498b16] initial commit
 4 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 document.csv
 create mode 100644 temp-average.py
 create mode 100644 temp-data.csv
 create mode 100644 temp-doc.txt
```

Read the response. It indicates that the commit was done to the **main** branch, and included four files.

The next step is to `push` to an online repository or a server. But we will do this later, with a repository first created online and copied to our system.

Create a repository in GitHub

We are going to create a repository in GitHub, creating a fork of an existing repository. Login to GitHub, go to https://github.com/isa-ulisboa/greends-fads-first-project and fork the repository. The fork of this repository will be added to your account.

After this, we can copy the repository to your local system, doing `clone`. First, lets choose the directory to where we want to clone the repository. In this case, it should be one level up to the directory *my-first-project* we created before:

```
$ pwd
/Users/rfigueira/my-first-project

$ cd ..

$ pwd
/Users/rfigueira
```

Go to GitHub and copy the link to clone the *greends-fads-first-project* we forked, and run in the terminal:

```
$ git clone https://github.com/isa-ulisboa/greends-fads-first-project.git
```

In your system, a new directory named *greends-fads-first-project* will be created.

Update your repository and push to GitHub

We will make updates to the repository we cloned and then push them to the original remote repository.

Let's change the current directory to inside *greends-fads-first-project*, and copy the files from the *my-first-project* of before:

```
$ cd greends-fads-first-project

$ pwd
/Users/rfigueira/greends-fads-first-project

$ cp ../my-first-project/* .

$ ls
README.md   temp-average.py temp-data.csv   temp-doc.txt

$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md
        temp-average.py
        temp-data.csv
        temp-doc.txt

nothing added to commit but untracked files present (use "git add" to
track)
```

Questions / Tasks

- Can you explain the copy command used above?
- Can you explain why, in this case, we did not have to do `git init`?
- Can you commit the files copied? What do you need to do? Execute the commands that are needed.

After this, the git status should give the following status:

```
$ git status
On branch main
Your branch is based on 'origin/main', but the upstream is gone.
```

```
  (use "git branch --unset-upstream" to fixup)

nothing to commit, working tree clean
```

Now, we can push our changes to the original repository. The generic command has the form `git push <remote> <branch>`. Let's apply it to our repository:

```
$ git push origin main
```

The terminal will ask for your GitHub username and password. In this case, the password is your Personal Access Token that you should create at your GitHub account at https://github.com/settings/tokens.

After the `push` command completes, go to your repository in the browser and confirm the changes.

Update the local repository with pull

If you are collaborating with others, there will be changes in the online repository that we want to bring to your local repository. To do this, we use the `pull` command:

```
$ git pull
Already up to date.
```

The response indicates that no changes occurred on the online repository in relation to the last push.

Pull request the fork to the original repository

Finally, we want to submit our changes in the fork to the original repository. This can be done at the repository's page at GitHub.

Wrap-up

In this exercise, we learned the very basics to use git and GitHub:

- what is version control
- config git
- use of `git status`
- create a new repository with `git init`
- the steps to update a repository: `add`, `commit`, `push`
- create or fork a repository in GitHub
- `clone` a repository
- update the local repository with `pull`

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