Introduction to Git

Goals

This guided exercise aims to be a Git short introduction. You are going to learn how to perform some basic tasks related to version control system. You are using Github to create a software repository.

Project overview

In this exercise you are going to work with a really simple python project. It contains 4 different files:

- mathtools.py: Simple and rather small python library which permits perform some basic math operations and the calculus of some series.
- test.py: Very basic test suite for the previous library
- README.md: Usually all projects have this file. It is used to explain what is all this project about. Majority of online version control systems present this file in the main page of the project
- .gitignore: We are not going in detail with this file in this exercise. Git uses this to define which files it should track. For instance, it is used to avoid that binary files are included in the repository. For instance, in the current project, the .gitignore file specifies that we should not track python binary files (.pyc)

You can access it from github at: https://github.com/ivanmilara/gittutorial You will download it later.

Step by step guide

Create exercise wiki.

As you did in Exercise 1, you need to create a webpage for the exercise:

1. Go to the course wiki page: 521002P Orientation to Computer Science and Engineering - Student area Home
2. Open your group page
3. Generate another Wiki page and link it to your document (inside "Link to exercise sessions" heading). **Use the template named "Version Control System Exercise"**. Name the page "Version Control System - Group nn"

    - Check that this page is under your group subtree.

Configuring Git in your computer.

NOTE: At least two members of the group must configure their git in their own accounts. So you would need to repeat the following steps in two different computers.

1. Open a windows shell (`cmd` or `GIT CMD`). It might be easier to use the `Git Bash` if you are familiar with linux commands
2. Type `git`. If everything is correct you should see the git help menu.

    Now we need to tell `git` your basic configuration settings. The command `git config` allows inserting the value of basic settings(email address, user name, ui format, default applications ...). Let's provide Git your name and email address. Git will use this information to identify you when sending a commit to the repository. Furthermore, we are going to tell git that we want to use colour when present text in the screen. We are also going to use notepad as our default git editor (different from the editor we use for writing code).

3. Execute the following commands in the GIT cmd:

Git configuration

```
git config --global user.name "<insert here your name>"
git config --global user.email <insert here your email address>
git config --global color.ui auto
git config --global core.editor notepad
git config --list
```

Creating project from Github

NOTE: INSTRUCTIONS IN THIS SECTION MUST BE COMPLETED ONLY BY ONE MEMBER OF THE TEAM

1. Log into Github (https://github.com/) and open your account.

    In Github you can create repositories from scratch (using the button "`New repository`". **DON'T DO THAT. In this case, you are going to**

FORK a pre-existing repository that we have created specifically for this exercise. A fork is a copy of the whole repository.

2. Once you have logged in into Github, open our project page https://github.com/ivanmilara/gittutorial and make a **`Fork`** of the project (button at the top right).

3. Only the members of your group should be able to update the the remote repository. Hence you **must modify the project settings** (see the tab located at the right with name **Settings**):
    a. Add all members of your team as collaborators. They should accept it in their email.
    b. If you want you can also rename the repo.

Cloning a project from Github

NOTE: INSTRUCTIONS IN THIS SECTION CAN BE DONE BY DIFFERENT MEMBERS OF THE TEAM IN DIFFERENT COMPUTERS

Once you have configured the project it is time to download a copy of the repo in your local machine. This operation is called **`clone`**.

HELP: You can find some help to solve this section at: https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-clone

1. Create a new folder in order to store the repository code. Name it `repo`. It should be a folder where you have read and write permissions (e.g. your HOME)
2. Open a windows shell (`cmd` or `GIT CMD`). It might be easier to use the `Git Bash` if you are familiar with linux commands.
3. In your terminal, move to the repo directory (use `cd` command). This is the place where we are cloning the repository.
4. Insert the adequate command to clone the repository. The address of the repository can be found from Github (`Clone or Download` button).
5. If everything went correctly, you should now see a directory named gittutorial (might be different if you have changed the name of your directory).
6. You can now open the different files of your project using your preferred text editor.

NOTE: You can also create a repository in your local machine and not in a remote repository. In that case you should use the **`git init`** command. **Do not use that one in this exercise.**

Creating and removing content from a repository

NOTE: INSTRUCTIONS IN THIS SECTION CAN BE DONE ONLY BY ONE MEMBER OF THE TEAM

Before going on, you must learn a really important command: **`git status`** (more info at https://www.atlassian.com/git/tutorials/inspecting-a-repository/git-status). This commands displays the state of the working directory and the stage area, that is, it lists which files are staged, unstaged and untracked. Furthermore, it tells you the branch in which you are currently working. You should use this command often, and it is a MUST before doing any commit, to check that the commit is correct.

1. Use **`git status`**, to check which is the current status of the project.

   Now you are ready to modify a file. Let's insert a new function in the **`mathools`** module to calculate the result of a factorial number. We want to store the changes in the repository

2. Open **`mathtools.py`** with a text editor.
3. Insert the following function definition at the end of the file. Please we careful with the white spaces.

```
def factorial(n):
    '''Returns the factorial of a number'''
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

4. Save the file
5. Commit the file to the repository. Some recommendations:
    a. You can find some help at: https://www.atlassian.com/git/tutorials/saving-changes . You can also check GIT CHEAT SHEET -> GIT BASICS.
    b. Remember that you must put the file in the stage area before committing it.
    c. Before committing a file, use the command **`git status`** to check that everything is correct.
    d. Remember to include a message for the commit. Use to that end **`git commit -m "message".`**
6. Check that you do not have any files to commit using **`git status.`**

Imagine that now you decide that you do not need the `test.py` anymore.

1. Use the **`git rm [file]`** command to remove the **`test.py`** file. Check that the file has been deleted from your working space.
2. Use **`git status`** to check that the file has been deleted**.**
3. Make a new commit to the repository (the **`rm`** command updates also the stage area, so do not need to use `add` this time)

Inspect a repository

NOTE: INSTRUCTIONS IN THIS SECTION CAN BE DONE ONLY BY ONE MEMBER OF THE TEAM

Many times you want to know who did a change in the code, when the change was made or what is the difference between two file versions. We have three different commands to that end (see the GIT CHEAT SHEET -> REVIEW HISTORY):

- **`git log`** (https://www.atlassian.com/git/tutorials/inspecting-a-repository/git-log): This commands lists the project history. If has multiple options that allows you filter and search for specific changes. It accepts a big list of arguments.
  - One interesting options is --oneline: it permits to reduce the information of each commit
  - When you are working with multiple branches --graph and --decorate are gonna be also useful
- **`git show <commit>`**: This commands shows the metadata of an specific commit.
- **`git diff <commit1> <commit2>`**: Show changes between two commits or between two branches.
  - Using **`git diff <commit1> <commit2> -- filename`** you can filter just one file

In this section, I invite you to play around with the above commands to answer the following questions in the wiki page of this exercise:

- **Q1: Using the `log` command determine the number of commits and their id (use just the five first digits) from the user named "Marta Cortés".**
- **Q2: Using the `show` command you must determine who, when and how many files were modified in the commit with id starting with `77d79` .**
- **Q3: Using the `diff` command you must determine the changes to the file README.md from the commit with id 4ee6d to the last commit (HEAD).**

Working with branches

NOTE: INSTRUCTIONS IN THIS SECTION CAN BE DONE ONLY BY ONE MEMBER OF THE TEAM

The main branch of development is always identified as **master**. You should not develop new features using master branch, but create a new development branch for each feature you are gonna implement. In this part of the exercise we are learning how to create branches and how to merge them to the **master.** Please, have a quick look to https://www.atlassian.com/git/tutorials/using-branches and the GIT CHEAT SHEET -> GROUP CHANGES where you can find more information about three different commands you will use when dealing with branches:

- **`git branch`**: Used to create and remove branches
- **`git checkout`**: Used to move your HEAD (the current state of the repository) to a different branch
- **`git merge`:** Used to integrate the content of one branch into another.

Now we are going to create a new branch in order to add a new function, named `fib to` the `mathtools` module.

Perform the following tasks:

1. Create a new branch named fibonacci.
2. Move to that branch and insert the code below into the **`mathtools.py`**. Before modifying the file be sure you are in the desired branch using **`git status`**.

```
def fib(n):
    ''' Calculates the n value of the fibonacci sequence'''
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1)+fib(n-2)
```

3. Commit your changes to the repository.
4. Now execute the command **`git log --oneline --graph --decorate`** and answer the following question:
   Q4: why HEAD is pointing to fibonacci and not to master?

Imagine now that you want to add a new function to the **master** branch. To perform this task:

1. Go back to the **`master`** branch. You will notice that the implementation for the function `fib` has disappeared (you have changed the branch!!!).
2. Insert the code from the following box into the **`mathtools.py.`** Before modifying the file be sure you are in the desired branch using **`git status`**.

```
def arithmetic(a, difference, n):
    '''Calculates the sum of a arithmetic serie of n elements.
       An arithmetic sequence is of the form: a, a+d, a+2d, a+3d,...
       n is the number of elements in the sequence.'''
    #Get the arithmetic sequence
    sequence = [a+difference*x for x in range(n)]
    #Calculates its sum
    return sum(sequence)
```

3. Commit your changes to the repository.
4. Execute the command **`git log --oneline --graph --decorate`**
5. Move again to the `fibonacci` branch and execute the same command as in previous step.

Q5: Take a screenshot of the command window both for the fibonacci branch and for the master branch.

You should notice that moving between branches modifies the content of the working directory. Now it is time to merge the content of the two branches into `master`.

1. Move again to the master branch. Check that you are in the desired branch using **`git status`**
2. Merge the fibonacci branch into master (**`git merge fibonacci`**)
3. Notice that git reports a Conflict. That is because we have modified the same file in two different branches.

Solving conflicts

NOTE: INSTRUCTIONS IN THIS SECTION CAN BE DONE ONLY BY ONE MEMBER OF THE TEAM

We need to solve the conflicts before we can make a successful merge. To solve a conflict:

1. Open the **`mathtools.py`** file (the file which generates the conflict)
2. Search for the conflict markers:
    a. Everything in between <<<<<<HEAD and ======= are the part of the file modified in the current branch (**`master`**)
    b. Everything in between ======== and >>>>>>> fibonacci are the changes made in the fibonacci branch
3. In this case we want BOTH functions in our file, so we just need to remove the markers: <<<<<<< HEAD, =======, >>>>>>> fibonacci
4. Save the file, add it to the stage (that will tell git that you have solve the conflicts) and make a new commit.
5. Execute the command **`git log --oneline --graph --decorate`**

Q6. Take a screenshot of the command window. Why the log shows two line in parallel?

Team collaboration. Working with remote repositories.

Till now you have always working in your local repository. But if you want to work in a team, you need to share the files in a remote repository. REMEMBER THAT YOU ALREADY CLONE THE REPOSITORY FROM GITHUB.

We have not yet upload our changes to our remote repository. Let's first upload the changes from our local repository into our remote repository:

1. Run the command **`git push origin`**. Origin is the nickname for the server of our remote repository.
2. Open your github project and check the number of branches. You should see only one branch (**`master`**). Why is that? Because git push origin just pushes the current branch. If you want to push other branch you should do it explicitly:
3. Run the command **`git push origin fibonacci`** in order to push also `fibonacci` branch.
4. Check in your github account that you have now two branches.

NOTE: REST OF INSTRUCTIONS IN THIS SECTION SHOULD BE DONE BY DIFFERENT MEMBERS OF THE TEAM IN DIFFERENT COMPUTERS.

Now that we have a remote repository up to date, we can work in teams. In order to identify two members of the team let's call them Alice and Bob. Alice is the one who has done all recent changes to the repo. Bob is going to start working in the repo now. A second member of the team must take Bob role's in a new computer. In order to start working in the project, Bob needs to create a local copy of the repository in its own computer. But **HE IS NOT GOING TO FORK A REPOSITORY, THAT IS, THE REMOTE REPOSITORY IS GOING TO BE THE SAME USED BY ALICE.**

1. Bob must follow the steps for Cloning a repository described in a previous section.
2. Bob should check that he has a correct version of the file.
3. **DO NOT CONTINUE TILL BOB HAS CLONED SUCCESSFULLY THE REPO**

Bob and Alice have decide that they are going to implement a new function, named `geometric`. This function calculates the sum of a geometric series of n elements. Both implement it differently:

Alice implementation

```
def geometric(a, ratio, n):
    '''Calculates the sum of a geometric serie of n elements.
       A geometric sequence is of the form: a, a*r, a*r*r, a*r*r*r,...
       n is the number of elements in the sequence.'''
    #Get the geometric sequence
    sequence = [a*(ratio**x) for x in range(n)]
    #Calculates its sum
    return sum(sequence)
```

Bob's implementation

```
def geometric(a, ratio, n):
    '''Calculates the sum of a geometric serie of n elements.
       A geometric sequence is of the form: a, a*r, a*r*r, a*r*r*r,...
       n is the number of elements in the sequence.'''
    #Use the sum formula:
    return a*(1-ratio**n)/(1-ratio)
```

Both Alice and Bob in, their respective computers are going to update the **`mathtools.py`**. They do not follow the recommendations provided by the teacher and both of them decide to implement it in their master branch. Bob and Alice must perform the following tasks.

1. Insert the code from above into the **`mathtools.py`** and save the file.
2. Commit the file to their local repository.
3. Both should update the remote repository (using **`git push origin`**), and their respective github accounts.
4. One of them should see a problem that does not allow him/her to upload the code. We need to solve the conflict before uploading the code. To that end, the person who received the error must DOWNLOAD the last version of the remote to the local repo.
5. Use **`git pull origin`** in order to get the last version of the repo. Usually, you should execute this command before starting working to update your repository.
6. Now you should see a merge **CONFLICT**. The version in the remote repository and your local repository has conflicts.
7. Open the **`mathtools.py`** file and locate the conflict.
   At this point Alice and Bob phone each other. They decide that the best way of proceeding is to use Bob's implementation because it is more efficient
8. Solve the conflict as explained in the section named Solving conflicts, using Bob's implementation.
9. Push the new version to the remote repository
10. Alice and Bob should now pull the content from the remote repository and check that everything is correct and they share the same commit number as last commit.

Q7. Take a screenshot of the Github commits page.

(OPTIONAL, ONLY DO THIS TASK IF YOU HAVE TIME) Checking and recovering old files and commits

Sometimes you want to check the status of a previous commit or even run its code. You can do that using the command:

- **`git checkout <commit>`**

Executing this command you move to a **detached HEAD state**. You can work normally, there (even do commits). But when you go back to master (**`git checkout master`**), all the changes you have done are lost. Hence this is only useful to explore previous version of the software.

Sometimes you want to substitute a single file of your working directory by a previous version of the file. You can do that using the command:

- **`git checkout <commit> -- <filename>`** (https://www.atlassian.com/git/tutorials/undoing-changes/git-checkout)

Using this information perform the following task:

1. Recover the file test.py that you removed in the beginning of the exercise. The file can be found from the commit with id `d444c`.
2. Save the file, commit it and push it to the repository.

(OPTIONAL, ONLY DO THIS TASK IF YOU HAVE TIME) Other Github tools

Github webpage offers multiple tools to interact with other projects. Pull request is a very useful tool that permit propose changes to an existing repository. Since you have made a lot useful modifications in your project, you can make a pull request to the original project (https://github.com/ivanmilara/gittutorial/). Following the instructions from the Github help page (https://help.github.com/articles/using-pull-requests/), send me a pull request.

test