Exercise 02 - Learning Git

Deadline

November 09, 2020 - 12:00/noon (UTC+1:00)

Submission Guidelines

1. Structure and work on your submission.
   - Create a directory (folder) for the exercise and name the directory ”exercise02”.
   - Place all relevant files/scripts for the homework in this newly made directory
2. Commit your work to the git repository.
3. Create a git tag for your repository when are ready to submit your homework
   - Tag name must be ”SubmissionExerciseSheet02”.
   - Tag can be made via the command line or using the GitLab GUI
   - Repository -> Tags -> New tag

Grading (10 pts):

Task	1	2	3	4	5	6	7	8	9	10
Points	1	0	1	0(+2)	1	0	0	2	3	2

Creating and Using a Repository

A Git repository is a specialized folder that tracks all changes made to files in your project. These tracked changes will build a history over time and you can trace every change made, to every file, back to the time and person who made them. This is incredibly useful for when you accidentally delete important code or if you want to retrieve previously written code that is no longer in your software. Using a repository to develop your code is standard practice by most programming teams and we will be using repositories extensively in this course.

Task 1 *Make your Own Repository* (1 pt)

1. *Create a Project*

- Starting at the homepage of GitLab, click on the New project in the top right
- Make sure you are in the "Blank project" tab and under Project name enter "PLAB2WS20-{GitLab Handle}" (replacing "GitLab Handle" with your GitLab account handle)
  - You should see the Project slug field change to look something like "plab2ws20-{gitlabhandle}", this will be the name of the folder later on
- Enter a description in the Project description (optional) box.
  - This repository will serve as the main location for your individual homework assignments. You can write up a simple description describing what will be in this repository
- Make sure you have "Private" selected under Visibility Level
- Make sure that the Initialize repository with a README box is checkmarked [x]
- Click Create project button

1. *Add Members*
   Once you have created your project, you will be the only one able to see what happens there. **Be sure to add the instructors to your project or they will not be able to grade your work**:

- Starting at the root page of your project, click on "Members" on the left side bar
- Under GitLab member or Email address, add "bschultz" and "jklein"
- Under Choose a role permission, choose "Maintainer"

- Click Invite

Task 2 *Clone your Repository* (0 pts)

At this point in it's life, your repo exists only on a remote server somewhere in the world. Though you can make and edit files in the browser, it is incredibly inefficient do so, especially for code. In this exercise, you will **clone** your repo to your laptop/desktop so that the files exist on your machine where you can edit them using whatever software you wish.

- Go to the repository's homepage in your internet browser
- Click the **Clone** button and select the copy URL button (clipboard symbol) under the **Clone with HTTPS** section
  - You can also clone using SSH if you have setup GitLab to recognize your SSH key. This is a quasi advanced feature that we will not cover here. If you would like to learn how to setup your computer to interface with GitLab without using login credentials, read the SSH documentation and the SSH Keys handout
- Open up your Terminal (macOS, Linux) or Git Bash Terminal (Windows) and navigate to the folder where you wish to store your repo folder
- Clone your repo using `git clone <pasted HTTPS URL>`
  - It will ask about your username and password. Be sure to input the same credentials that you used to login to GitLab
- You should see it download and afterwards check to see if your folder is there. It should be named the same as your **Project slug** from Task 1

Task 3 *Stage - Commit - Push* (1 pt)

1. *Edit the README* (1 pt)
   Now that your repo is cloned onto your local machine (laptop/desktop), you can make edits to files and **push** them to the remote repository (your repo that exists on GitLab). We will start by modifying the README.md file in your project. A README is a file that can contain many different things. It can describe what is in the current folder, explain how to use the software, provide links to resources, and much more. For this exercise, you need to modify the README and push it to the remote repo on GitLab.

- Now that your repo is cloned to your machine, open the **README.md** file in your project folder with a text editor (Atom, sublime, notepad) or IDE (PyCharm, VSCode)
  - If your project folder does not contain a README.md file, the you forgot to click the **Initialize repository with a README** in Part 1. Simply create a new README.md file in your project folder if this is the case
- Create a new header in the README.md file (a header is a line of text that starts with "#") that says "My README edit"
  - If you are using Atom or an IDE, you can visualize how your markdown file will be rendered.
- Save the file
- Using either the Terminal or integrated software, commit your changes and push it to your remote repo
  - If using the Terminal / command line, checkout out the Git commands handout
- Verify that your changes have been pushed by checking your repo in GitLab via your browser

(BONUS) Task 4 *Master Markdown* (+2 pts)

Markdown (.md) and ReStructured Text (.rst) files are the standard for outlining information pertaining to coding projects and technical packages. Learning the basic syntax can help you to better annotate your projects. Additionally, Jupyter Notebooks support Markdown, which makes it very useful for describing the different steps in your notebook so that others can follow your code and thought process. You can use markdown to create ordered (numbered) and unordered (bulleted) lists, tables, hyperlinks, footnotes, and much more. You can find a list of general things that you can do with markdown in our Markdown Handout and what special things GitLab allows with markdown in our GitLab Markdown Handout.

- Try to include as many markdown supported functionalities in your README

Task 5 *Issues* (1 pt)

1. *Submitting an Issue* (1 pt)
   Issues are an important tool for managing tasks and development of projects. In this course, students will use issues for submitting questions on homework, technical difficulties, and other inquiries. Students are encouraged to use the Issue tool

in GitLab for tracking what tasks need to be done, bugs in their code that need to be addressed, and (later on) assigning tasks to teammates. For this exercise, we simply want to see you create an issue and assign it.

- Starting at the root page of **your homework project**, click **Issues** on the left side bar
- Click **New Issue**
- Under **Title**, enter "Exercise 2 - Issue"
- In the **Description**, create a **task list** (checkmark/box symbol top right) and enter "Assign to instructor"
    - It should look like this in you **Description** box: `- [ ] Assign to instructor`
- Under **Assignee**, find "Bruce Schultz" and assign it to him
    - If you cannot find him, make sure he is added as a "Member" to your project (see Part 1b)
- Click **Submit issue** button on bottom
- After the issue is generated, click the checkbox next to "Assign to instructor" in the description of your issue

(BONUS) Task 6 *Learn the Issue Tool in GitLab* (+0 pts)

The issue tool has many features including a Kanban Board and Labels.

Kanban Boards are used to visualize at what stage of development an issue is at (e.g. assigned, being worked on, finished (closed), etc). GitLab has a default board you can generate (Issues -> Board) which has 4 stages: Open, ToDo, Doing, Closed.

Labels are used to add additional annotations to issues to better filter/separate them. Labels can be made to specify whether an issue belongs to a particular project, refers to a bug or feature of the software, and so on.

- Play around with the Issue tool on GitLab. Try to generate your Kanban Board and some labels that you might use for annotating your issues.

Virtual Environments

Virtual environments are an incredibly useful tool for developing code and building software in an isolated workspace.

Task 7 *Start Working in a Virtual Environment* (0 pts)

While this task is not worth any points because we cannot enforce or verify it, it is **HIGHLY** recommended that you learn how to do this now as it will prevent a lot of problems later on when we start using more complicated packages and for the group project.

1. *Create a Virtual Environment (venv)* (0 pts)

- Create a new virtual environment named "homework" (or whatever you would like)
    - For more information on how to do this, see the information handout
- Activate your virtual environment (*HINT* if you are performing this step in a terminal, you should see "homework" in parantheses at the beginning of your input line)

1. *Install networkx in your venv* (0 pts)

- Install networkx in your virtual environment
    - To test that you are in your venv, use `pip list`. You should only see a couple of packages including `networkx`

Task 8 *Compile Node / Edge Lists from PPIs* (2 pts)

Given a list of protein-protein interactions (PPIs), generate an edge list and a node list.

- Write a function that is able to read the provided PPI text file and output a node list file and an edge list file
    - A *node list* is a file that maps unique identifiers to a node. In this exercise, your node list should be a **tab** separated (.tsv) file in which each line consists of two values: a unique ID (such as an integer) and the HGNC symbol
    - You will need to develop your own method for assigning **unique** identifiers to each HGNC symbol
    - An *edge list* is a file in which each line contains a pair of node identifiers (indicating an interaction between them) and edge attribute information. Again, this should be a **tab** separated file.

    - An edge list should use the node identifiers, NOT the node names/symbols.
    - Additional columns should be included in the edge list to include edge types

Examples

Node List

```
1 CD33
2 CXCR5
3 IFNG
```

Edge List

```
23  45  {metadata}
98  21  {metadata}
31  16  {metadata}
```

Task 9 *Depicting the Network* (3 pts)

While NetworkX's primary goal is to generate and analyze graphs, it currently is capable of outputting images of the generated graphs using other packages such as matplotlib and pygraphviz.

1. *Draw It* (1 pt)

- Write a method that reads your generated edge list file and compiles a graph from it.
    - You may need to install `matplotlib` and `scipy` in your venv

1. *Style It* (1 pt)

- Modify your function so that the generated graph image includes:
    - HGNC symbols next to the nodes
    - The nodes are colored 'red'
    - The graph positions the nodes according to the Fruchterman-Reingold force-directed algorithm (also know as "spring" layout)

1. *Export It* (1 pt)

- Write a function that generates a PDF of your graph

Task 10 *Command Line Compatible* (2 pts)

Scripts should be executable from the command line whenever possible, this allows users to quickly execute code without having to open up additional software.

- Design your script (.py file) so that if it is given a list of PPI interactions (such as `exercise02/data/ppis.csv`), it automatically compiles a network and outputs an image of the network to a specified location.
    - Your command line interface should accept two parameters:
    - File path for a PPI CSV and
    - file path for the image output
    - Example command that should work: `python3 my_networkx_script.py /home/myname/Documents/ppis.csv /home/myname/Desktop/ppi_network.pdf`