Basic GitHub workflow

We will go through an example of standard tasks you would need to perform in relation to Git and GitHub.

Notes

Creating issues

We will perform the following steps:

- Create a new issue in GitHub
- Practice with things we can do with that issue (labels, assign, close, reopen)
- Changing some code and viewing the changes we made.
- Making a commit to automatically close the issue.

First off, let us make an issue. Open the GitHub project in YOUR account that is called WebAppsLabs. Typically just going to github.com[1] will automatically take you to your starting page. You should see a "repositories" section, with "WebAppsLabs" in it, click that link.

For this time only, we need to enable the "issues" tracker for your project. To do this, choose the "Settings" section on the right, and check the "Issues" checkbox in the "Features" section.

You should now see a new icon on the right, the Issues icon that looks like an exclamation mark, so click on that. You should see a very welcoming message.

Let's create a new issue! Click the "New Issue" button on the top right.

- For "title" put "Add our name to yourCode.js".
- From the Labels section on the right chooose "enhancement" and "bug".
- Right below the title it says "no one is assigned". Choose to assign it to you.
- You can leave a comment in the box right below the "Write" message. Leave it empty for now though. Click on "Submit new Issue".
- You should now be looking at the issue. You see that it has the tracking number 1, and along the way it says that two labels were added to it, and it was assigned to you.
- Leave a comment! Type the following in the comment space (made up to allow you to learn some of the Markdown syntax)

[1] https://github.com

```
This is my **first** comment! I *love* it! I'm going to make a list of points about it:

- A first item
- A second item

    1. Move four spaces in to create a "subitem"
    2. A numbered list, woohoo!

- I will use `backticks for code`

```
I will use "fenced by 3 backticks" blocks for whole code blocks
```
- I can create a "mention" to another user by typing an at-sign: @skiadas to draw their
- This will actually make that user receive emails, so you've just bombarded your instru
```

After you have typed all that in, click the "Comment" button. This should add a comment, and offer you a space to leave another comment. Look at how the things we typed end up looking! That's Markdown.

- Go ahead and type another comment in, but this time click the "Close and comment" button instead. You will notice that this marked the issues as "closed" meaning "resolved", and it also says who closed it and when. Let's go ahead and use the "Reopen Issue" button to reopen it.

  By the way, remember how you got the comment earlier to send an annoying email to that skiadas fella? Well, he also received a notification for each one of these changes you just made, oh what joy!

  But the point is that issues can be a very effective place to have a conversation with someone on an issue.

- You can also change the labels of an issue after it has been assigned, go ahead and do that (more emails for @skiadas, yey!).

- After you've had enough fun, click back at the exclamation point to the right, and you should see a list of all "open" issues, only one in this case. You see the title for it, the labels attached to it, when it was opened, how many comments it has and who is assigned to it on the right side.

Making Commits

And these are the basics of creating issues. We will now make a change in our code.

- Make sure the Terminal is pointing at the WebAppsLabs directory, use the "`cd .....`" command to navigate there.

- Open up Sublime Text with "`subl .`" or in any other way you like, or whatever editor you like, and find the file `yourCode.js`. You will see that the first couple of lines have room to put the names of the two people in the project. Go ahead and edit those two lines, then save the file. Then go back to the Terminal.

- If you do `git status`, it should now show you that there is one file with modifications, and that you are in the Lab1 branch.
- Now do `git diff yourCode.js` to see the modifications you have made. Don't forget to use "Tab" to autocomplete the name.

  The lines with a dash in front of them will be removed, the ones with a plus will be added.

  If you want to see the diffs in your editor instead, do "`git diff yourCode.js | subl`".
- Looks good! So go ahead and add that file, with `git add yourCode.js`.
- Do a `git status` now. You should see it as "to be committed".
- Remember that while `git diff yourCode.js` will not show you anything now, as the changes have been staged, you can see the staged changes with `git diff --staged yourCode.js`.
- Now do a `git commit`. For a message, put "`Add our names. Close #1`"
- It is important that you spell the last part correctly. You can however type "Fix" instead of "Close". Save and close the commit file.
- Now if you do `git status`, it should tell you that there is nothing to commit. It also says something interesting: Your branch is ahead of `'origin/Lab1'` by 1 commit. That is the name of the Lab1 branch that is stored on GitHub's servers. The commit we did was a local commit. We now need to push our changes to the remote.
- So do `git push`. It will ask you for your GitHub login and password. When you are done, open the web browser back up and look at the issues page now. You should see that your issue is now closed, and it even links to the commit that did it. This is because of the "Close #1" we put as part of the commit message. GitHub automatically picked it up.
- NOTE: It might actually not be automatically closed. But you will definitely see the issue now mentions the commit we just did.
- Try to always include issue numbers to the commits made to refer to them.