Digital Technologies

Lab 2 - Working with Git

Task 1 - Create a GitHub (or other) account

The first task is for us to create an account on a public Git server. This is a convenience so that we don't need to run our own server. It also has benefits for use as a portfolio.

- Navigate to https://github.com and select sign up
- You might want to use a personal email

With this done, we will now create a new repository using the web interface. Just remember to "tick" the box that mentions initializing the repository with a README file. This means we do not need to initialize the repository manually.

Once you have done this, make sure to check out the new repository on your local machine with the following command:

git clone https://github.com/<yourusername>/<reponame>.git

Task 2 - Download an example project

Our next task is to download a small example project containing a little bit of code and images to use as a placeholder so we can get used to how it interacts with Git.

The following command will download a zip file containing the sample project:

wget https://github.com/osen/openday/archive/master.zip

Once downloaded simply extract the .zip somewhere in your home directory and then move the files into your repository that we created earlier.

Task 3 - Add the initial files to Git

If you run the following command:

git status

You will see that Git has detected the new files, however they are not yet part of the repo and they are not staged. We need to stage them by running:

git add .

This command will add all files in the current directory (remember the hidden files . and .. are directories). Then we can commit the changes to our local database with something like:

git commit -m "Initial import of some sample files"

And finally this local database needs to be merged with the servers running GitHub. This can be done with the following command. Note that it will ask your username and password which will not echo so just keep typing, it is working.

git push

If you refresh your GitHub repo page at this point, you should see the newly added files.

Task 4 - Compile and make changes

So lets build and run this project (to create some files we don't want to version). This can be done by making sure to cd into the project directory and using the Makefile:

cd <projdir>
make

You can either run the executable by double clicking on it in the project folder, or you can run:

./openday

If you obtain an overview of the project now by running:

git status

You will see a bunch of files in red that you do not want to add to version control. This is because these change very often and can be generated from the source code.

Task 5 - Making changes

From the project directory if you launch Notepad++ via:

npp

You will see the project files and can edit them. Make a small change within the *src/main.cpp* file such as changing the window title (this is on line 41) and rebuild with **make**. While we are at it, lets also change the image slightly in *resources/player.png* (Perhaps remove his mustache and the M from his hat to ensure our game is more compliant with intellectual property!).

Finally run list the working directory again with:

git status

Now it is up to you to decide which files should be added. Which of these files (in red) do we want to version? Imagine you were a new contributor to the project and needed to download the initial project files. Which ones would you need?

Hopefully you will notice that all the files in src and resources will be the ones that need committing. In that case we can simply run:

git add src resources

This will add any changes in just those two folders and ignore the rest. This can be confirmed with:

git status

And now we simply commit the work and push it:

git commit -m "Changed title, updated character"
git push

So as you should hopefully see by this task, it is fairly easy to make changes and commit your work. We only really needed three git commands (add, commit, push).

Task 6 - Making conflicts

Now lets try to make a conflict. Open up another command prompt (or use tmux) and clone the project elsewhere into the *other* directory:

git clone https://github.com/<yourusername>/<reponame>.git other

Now, could you edit the game source code to change the title again on both working copies. After adding, committing and pushing for each, you should arrive at a merge conflict. It should look similar to the following:

CONFLICT (content): Merge conflict in src/main.cpp
Automatic merge failed; fix conflicts and then commit the result.

So for code, this is usually slightly less common. For example if two different lines are modified Git can automatically merge them. However because in this lab we edit the exact same line, we arrive at this.

Open the file in Notepad++ and have a look through the file for the merge markers. Simply remove them from the file (and deleting one of the lines we do not need). When you are happy, double check things with:

git status

Instruct Git that you believe you have solved the merge by running:

git add src/main.cpp

Note: If Git detects any merge markers still in the file, it will notify you and ignore your add command.

Finally run the following to commit and push your merge (just like any other change):

git commit -m "Merged title changes"
git push

Task 7 - Creating binary conflicts

Now lets do the same for the image. In both Git folders, make a change to the player image and run through the add, commit, push steps.

Just like before you will arrive at a merge conflict. This time it is because binary files cannot be merged.

warning: Cannot merge binary files: HEAD:player.png vs. player.png

In some ways this is easier. You either have the choice to use "yours" or "theirs". To use yours, simply add and commit the file again.

git add resources/player.png

However, lets use "theirs" for now. In this case all we are doing is re-checking out the latest revision of the file (i.e their one), then add the change (because it is still changing your project).

git checkout master resources/player.png
git add resources/player.png

Finally, regardless of if you went with theirs or kept yours, you want to commit and push:

git commit -m "Merged player changes"
git push

Task 8 - Group work

Lets split into groups of 2-4 and practice working on the same project. In the GitHub web interface, click on settings and you can add collaborators. Try doing this now.

- First start by one person making changes and the others using pull to receive it.
- Try changing different lines each and committing.
- Try all changing the same line and committing.

It may be confusing at first but it will be a useful thing to practice when it comes to the group project unit in the second year.