CSCI 1010 Computer Science Orientation
Introduction to version control

Our activity for today Our activity for today will focus on the following learning objectives:

- Getting experience editing html.
- Getting experience with the command line.
- Getting experience with version control and cloning, committing, and resolving conflicts in git.

1 Why version control?

Version control is an immensely valuable tool for programming (in addition to many other collaborative activities, like writing). It allows you:

- Safely collaborate on projects
- Backup your work easily
- Revert broken programming code to a previous state
- Maintain multiple versions of a project
- Experiment with implementing new features without overwriting working code

2 Install git

OS X On OS X, go to **https://git-scm.com/download/mac.**
Windows On windows, go to **https://git-scm.com/download/win.**

3 Test git

In both cases, you will test git through the command line (its possible to interface to git through your preferred development environment, but I will not cover that here). In OS X, you will launch the command line by running Terminal. In Windows, you will launch the command line by running a Command Program Window.

```
$ git --version
```

And you should see something like:

```
git version 2.11.0
```

4 Clone the repository

Cloning a repository means that you make a copy of the central repository. You do this the same way on OS X and Windows:

```
$ git clone https://github.com/GWU-CS1010-F17/Students-Fall18.git
```

You will see something like this:

```
Cloning into 'Students-Fall18'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100\% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100\% (3/3), done.
```

If you mis-typed the name of the repository you will see something like:

```
Cloning into ‘Student-Fall18’...
remote: Repository not found.
fatal: repository 'https://github.com/GWU-CS1010-F17/Student-Fall18.git/’ not found
```

5 Making your changes

Your task is to add your student information (picture, name, hometown, and hobbies) to the table in the web page students.html, and to add your name to your team in the webpage teams.html located in your newly created Students-Fall18 directory.

Important requirement For the students page: there must be exactly three students in each row, except for the last row, which can have one, two, or three students. For the teams page: each team has to have a name, and a name for their robot. There must be exactly five students in each team, with the exception of two teams that are allowed to have for students.

Adding your picture: Copy your picture to the “imgs” directory and add it to the repository. If my picture is called roxana.jpg, this would be the process I would go through to add it:

```
$ cp  ~/Downloads/roxana.jpg imgs/roxana.jpg
$ git add imgs/roxana.jpg
```

This last line is very important. It will be responsible for copying the file from your local repository to the central repository.

5.1 A little about html

Dont be afraid to experiment. You wont break anything!

The first thing you should know about html is how to write a comment:

```
<!-- everything within the brackets is ignored -->
```

The next thing you should know about html is that an open element should be followed by a close element, like this:

```
<p>
<!-- everything within the brackets is ignored -->
</p>
```

Above, the open element is “<p>” and the close element is “</p>”. A description of the html elements that you will be modifying follows:

table Your web browser will render elements in tables in nice, orderly rows and columns. You dont have to explicitly specify the numbers of rows and columns in the table.

tr A row of elements in a table.

td A single element in a table.

p A paragraph of text in a web page- your web browser separates text and images in paragraphs apart from text and images in other paragraphs.

img Embeds a picture in a web page. src points to the image filename.

ul This sets up an unordered (bulleted) list.

li An item in a list (ordered or unordered).

5.2 Viewing the web page

Want to see what the web page looks like with your changes? Open the file students.html in your web browser. Its that easy. subsectionWhat if you screwed the web page up badly? If your changes really screwed things up so badly you dont know how to fix it, dont worry. You can always do this:

```
$ git checkout -- students.html
```

This reverts your changes to the file.

6 Telling git who you are

Before you do anything else, tell git who you are (Make sure you replace the email address and name below with your email address and name):

```
$ git config --global user.email "you@example.com"
$ git config --global user.name "Your Name"
```

7 Committing and pushing your changes

Once you are satisfied with your changes, you want to "commit" them. I like to think of a commit as a bookmark. It says to git that "Im happy with saving my work right here."

REPLACE THE TEXT BELOW WITH A UNIQUE COMMIT MESSAGE.

```
$ git commit -m "Added my info" students.html
```

Pro tip: only commit programming code that compiles.

7.1 Pushing your changes

You can commit many different changes without the central repository being updated. The central repository is only updated when you "push" your changes. Pushing will be done via the following:

```
$ git push origin master
```

You will execute this command, accepting for the moment that it is a bit of magic. I recommend looking at online tutorials on git if you want to delve into this command further.

7.2 Resolving conflicts

If all went according to plan you will see something like this:

```
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 307 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/GWU-CS1010-F17/Students-Fall18.git
55742e1..d7fb909 master -> master
```

7.2.1 Conflict resolution: easy

By the nature of this exerciseall students editing this web page simultaneouslyit is highly unlikely that all did go well. You are then likely to see something like the following message:

```
To git@github.com:GWU-CS1010-F17/Students-Fall18.git
 ! [rejected] master -> master (non-fast-forward)
 error: failed to push some refs to 'git@github.com:GWU-CS1010-F17/Students-Fall18.git'
 hint: Updates were rejected because a pushed branch tip is behind its remote
 hint: counterpart. Check out this branch and merge the remote changes
 hint: (e.g. 'git pull') before pushing again.
 hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

This means that someone pushed before you, meaning that your edits are based on an old version. git will do its best to resolve this for you if you "pull" then "push". In such cases (automatic conflict resolution), the result will look something like this:

```
$ git pull
Auto-merging students.html
```

You can than do:

```
$ git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 307 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/GWU-CS1010-F17/Students-Fall18.git
55742e1..d7fb909 master -> master
```

This message indicates a successful result.

7.2.2 Resolving conflicts: hard

If the automatic conflict resolution fails, youll see a message like this during the pull:

```
$ git pull
Auto-merging students.html
CONFLICT (content): Merge conflict in students.html
Automatic merge failed; fix conflicts and then commit the result.
```

This means that git wasnt intelligent enough to solve the problem on its own: you have to do it. git provides many options to do this: you can use the version currently in the repository ("theirs"), your current version ("mine"), or manually resolve the conflicts in the file.

In our case, part of "students.html" will look something like this:

```
              <!-- commented out, b/c DJT isnt really part of our class
              <td>
                     <p>
                     <img src="imgs/trump.jpg" height="200" />
                     <ul>
                            <li>Name: Donald J. Trump</li>
                            <li>Hometown: NY, NY</li>
                            <li>Hobbies: Golf, Self-aggrandizement</li>
                     </ul>
                     </p>
              </td> -->
<<<<<<< HEAD
              <td>
                     <p>
                     <img src="imgs/jane-doe.jpg" height="200" />
                     <ul>
                            <li>Name: Jane Doe</li>
                            <li>Hometown: Los Angeles, CA</li>
                            <li>Hobbies: ping pong, foreign languages</li>
                     </ul>
                     </p>
              </td>
========
              <td>
                     <p>
                     <img src="imgs/john-doe.jpg" height="200" />
                     <ul>
                            <li>Name: John Doe</li>
                            <li>Hometown: Kalamazoo, MI</li>
                            <li>Hobbies: practical jokes</li>
                     </ul>
                     </p>
               </td>
   >>>>>>>> master
       </tr>
</table>
```

The part of the file within "<<<<<<< HEAD" and "========" are the changes you (Jane Doe) just made. The part of the file within "=======" and ">>>>>>> master" are the changes the other student (John Doe) just made. It is these two parts that git does not know how to combine together. Here is how I would resolve the conflict assuming that I wouldnt have to take any special actions to keep *two students to a row:*

```
                <!-- commented out, b/c DJT isnt really part of our class
                <td>
                        <p>
                        <img src="imgs/trump.jpg" height="200" />
                        <ul>
                                <li>Name: Donald J. Trump</li>
                                <li>Hometown: NY, NY</li>
                                <li>Hobbies: Golf, Self-aggrandizement</li>
                        </ul>
                        </p>
                </td> -->
                <td>
                        <p>
                        <img src="imgs/jane-doe.jpg" height="200" />
                        <ul>
                                <li>Name: Jane Doe</li>
                                <li>Hometown: Los Angeles, CA</li>
                                <li>Hobbies: ping pong, foreign languages</li>
                        </ul>
                        </p>
                </td>
                <td>
                        <p>
                        <img src="imgs/john-doe.jpg" height="200" />
                        <ul>
                                <li>Name: John Doe</li>
                                <li>Hometown: Kalamazoo, MI</li>
                                <li>Hobbies: practical jokes</li>
                        </ul>
                        </p>
                 </td>
        </tr>
</table>
```

Once Ive completed this, I now take a look at the web page again to make sure that it looks ok. **Do not skip this step: your grade will suffer if you break the page!** Assuming that the web page looks good, commit and push:

```
$ git commit -m "Fixed merge conflict" student.html
$ git push
```

In the worst case scenario, someone else also tried committing, got a merge conflict, and corrected it and pushed in the time you were trying to correct your conflict. In that case, youll have to repeat the steps at the beginning of this section all over!

7.3 Viewing the live web page

After you have successfully pushed your changes, you will shortly (within a minute or so) be able to view your changes to the live web page at:

https://gwu-cs1010-f17.github.io/Students-Fall18/

8 Assessing your grade

Your grade will be assessed in the following manner:

10% adding your picture to the repository

15% adding your info to students.html

15% adding your info to teams.html

20% resolving merge conflicts

20% ensuring that there are only three students per row

20% verifying that the web page is correctno typos, no weird charactersbefore you push