GitHub, D3 and Data

In this exercise we are going to look at how we can use Git to manage and visualise data. Although designed to manage software development, Git can also be used to manage any type of resources that require complex policies to be applied to carefully control the quality of the material. GitHub is a free to use online Git platform for storing Git pojects.

Benefits of Source Control

Diffs: The history of changes, who made them and when. Linked to comments, bugs and milestones, differences also give you the context of why.

Branching: Allows you to make changes to a project without affecting the “master” version. This technique can be used to trial new ideas, fix bugs or handle task management.

Merging: Is the operation of combining branches (and thus the changes) together. This is often performed between a branch and the “master”. Advantages of making changes on branches prior to merging include allowing of quality control processes to be applied.

Conflict Resolution: While merging and committing, you might find that someone else has changed the same thing as you, causing a problem. In some cases you can merge, in other cases you may have to open discussion to resolve the problem.

Tagging: A critical part of the release stage of a product. Whenever a formal release is made, tagging helps you track at exactly what point, and with what parts, a release was made. This way if a bug is reported against a release, you have a way to get back to that exact release.

Bug Tracking: The ticket tracking system for version control. Allows you to keep track of issues with your product, set targets and assign people to tasks. Additionally, in the majority of version control systems, bugs can be closed with commit messages and from branches.

Milestones: Typically used for release management, but can also be used for more generic wish lists. Milestones should always be combined with the bug tracking system in order to track progress on tasks.

Wish lists: A different take on bug tracking and milestones.

Exercise

This exercise will introduce you to the notation of collaboratively editing and managing an open data project using Git. Although this exercise should be done programmatically in a live environment, we are going to use the GitHub web interface to manage our repository.

Each member of the team will need a GitHub account, available at http://www.github.com.

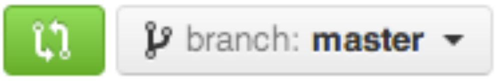
Step 1 – Exploring the Repository

For the purposes of this exercise, an example dataset has been started at the following location (where X is the number written on the front page of this exercise:

https://github.com/ukodi-training/ODP-GroupX

Opening a web browser at this location should show you the main GitHub screen with the "Code" tab highlighted on the right hand side. You will notice that there are two files, a README.md and a LICENSE. Feel free to click on both of these files to take a look at their contents, note that the README is displayed by default at the bottom on the code tab.

Clicking on the network tab will reveal the changes graph for this repository. It will be noticeable that there is not much activity on this repository, however there are two branches called **master** and **gh-pages**. From the code screen (back), select the **gh-pages** branch from the branches dropdown.



While branches are usually used to store work in progress copies of the code (one for each feature or bug being added or fixed), the gh-pages branch is special as it stores a web site related to the master branch. As our code is a web site with visualisations we may as well store it in gh-pages. To access the web site for your group in a new browser tab open the following location:

http://ukodi-training.github.io/ODP-GroupX/

This shows how it is possible to use Github to host an entire website, in this case hosting a completed copy of the TFL Tube status code from the earlier exercise as well as the beginnings of a new site that will contain a shaded world map.

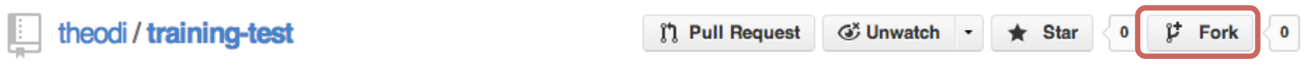
Going back to **Github**, the other tab of note is the issues tab. Here you will note a whole list of issues outlining the tasks that need to be completed as part of this exercise. Additionally there are a number of milestones that relate to each section in the exercise, one for visualisation and one for interaction.

In this exercise we are going to create our world map visualisation and each group will create the code, ticking off issues and milestones as we go. As it is not possible to directly edit the projects owned by ukodi-training, you will first need to create your own development copy, then edit the code before submitting requests back to ukodi-training to merge your changes. This process will demonstrate how a community can contribute to managing data and code via a collaborative platform, showing how full historical information will be stored automatically.

Step 2 – Fork the Repository

Only one person in each group should do this. If more than one person in the group wishes to make changes to your repository then you should set up collaborators in the settings menu once you have forked the repository.

As you will not have direct privileges to edit the master repository, you will need to create your own version that you can work on independently of this version. To do this, you will need to find the **fork repository** button on the training-data repository page.



Once you have forked the repository you will be re-directed to a location where you are free to edit the files without danger of affecting the master version.

Once you have your local version you now need to add all your team members to this repository as collaborators. To do this click the **settings** tab and then find the **collaborators** section. You will then need to add each member of the team by their GitHub username or other identifier.

Step 3 – Getting started

Once everyone has access to the repository you have set up, you have now formed a development team. All of our development is going to be done against the **gh-pages** branch so make sure you switch to this straight away. You should also be able to see your own version of the website at the following URL:

http://*your-username*.github.io/ODP-GroupX/

You will note that Issues and Milestones are not copied when you take a fork of an existing repository. In order to follow the exercise you should keep a copy of the ukodi-training source repository open so you can view these.

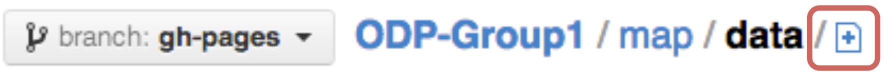
The next steps will take you through how to fix each issue and complete each milestone. At all times we will be working in the **map** directory.

Step 4 – Adding Data

The first stage is to add data to visualise to the map/data directory. While any csv file that contains a column (named **country**) listing country names can be used, there is an example one on the course website that contains global population data.

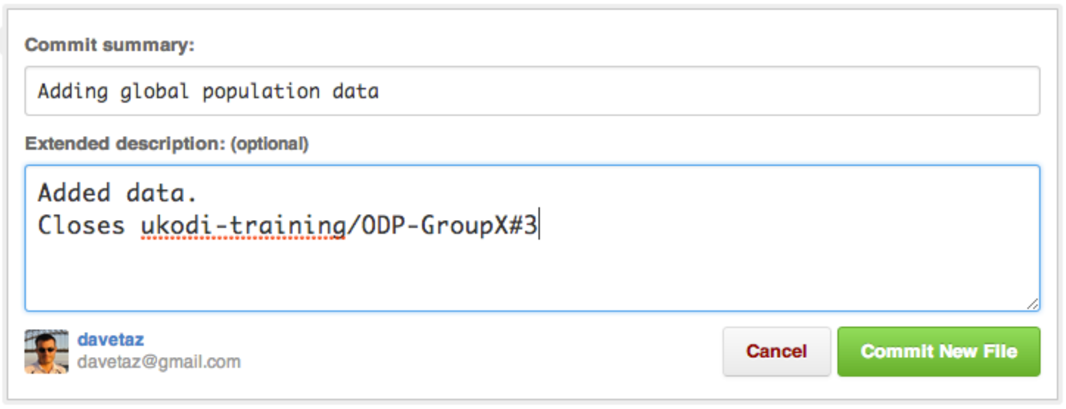
In order to add this file to the github repository you will first need to open it in a text editor (so you can see the csv) and then copy the contents to your clipboard.

In github browse to the data directory under the map path and you will see a plus button that allows you to create a file via the web interface.



Make sure you give the file a **name and extension** (e.g. data.csv) (needed in step 5) and then paste the contents of the file in the edit box shown.

Once you are done adding your data you now need to commit your change. To do this you are required to complete a commit message detailing your change. You can change the summary and description, however at some point in the **description** you **MUST** include a link to the issue you are closing in the master "ukodi-training/ODP-GroupX" repository. The reason for this will become clear later in the exercise. Below is an example of a very simple commit message that is the proposed close to issue #3.


Commit summary:

```
Adding global population data
```

Extended description: (optional)

```
Added data.
Closes ukodi-training/ODP-GroupX#3
```

davetaz
davetaz@gmail.com

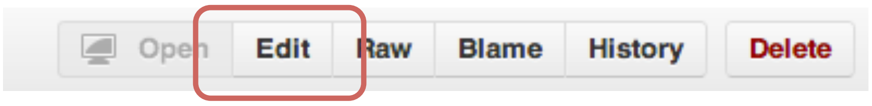
When done simply click "Commit Changes" (in this case "Commit New File").

Step 5 – Adding a map section

In order to display the map on our web page we need to make a section in which it is displayed. To do this we need to add the following somewhere in the main body of the main web page in the map directory, **index.html**.

```
<section id="map"></section>
```

To edit this file, click it in order to display the file contents and then click the edit button.



Having previously explored HTML to create the TFL tube status page, this page should not look that much more complicated. Note that a completed example of the TFL exercise can be found in the tfl_tube directory.

Once again you will need to commit your changes and add the correct issue number to the comment that you are closing.

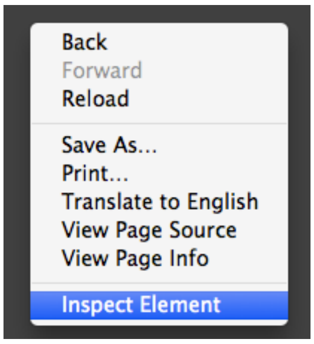
Step 6 – Making the map display

For the purposes of this exercise a simple configuration file (**js/config.js**) has been created which allows the customisation of a number of aspects of the map. In this file you need to define the name of the **data file** you created in step 4 and also the title of the column that contains the numbers (e.g. **Population**).

Once changes have been made, commit your file, closing the relevant issue and then browse to your website to see if it all worked.

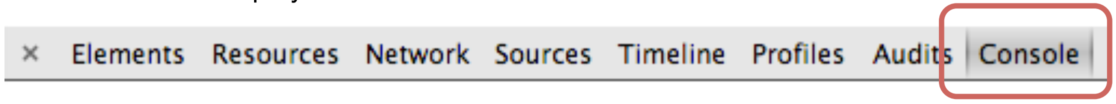
Step 7 – Debugging

If you managed to show a map, fantastic! If not then this section is for you.



Many web browsers come with nifty tools to help debug errors on a web site, for the purposes of this exercise we are going to use the **Web Inspector** in **Google Chrome**. To access this first ensure you have your broken web page displayed and then right click anywhere on the page itself and click **Inspect Element.**

The web inspector is a very powerful tool and allows you to view the source of a web page, view its performance as well as browse locally stored data. The feature we are interested in is the console, where errors in our scripts will be displayed.



In the case of this exercise some of the errors you might see may include:

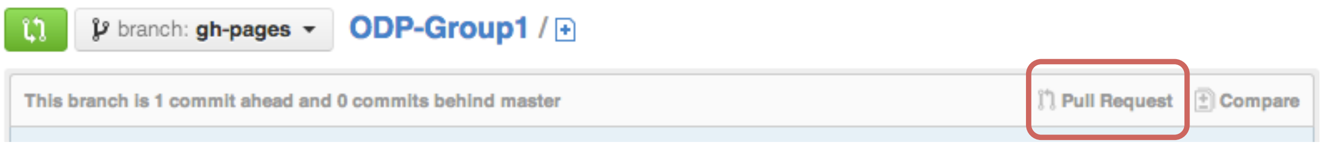
- Failed to load resource: the server responded with a status of 404 (Not Found)
  - Probably an error in your config or the data file is missing
- Failed to parse, unexpected …
  - You forgot to close a bracket or have too many/few commas and semi-colons in your config file.
- Uncaught typeError
  - The data file is not a csv?

If the map displays but it not shaded then it probably didn't find data for any known country name. This means that the country column is missing or uses strange non standard names for countries which are unknown to the map.

Step 8 – Sending Pull Requests (Part 1)

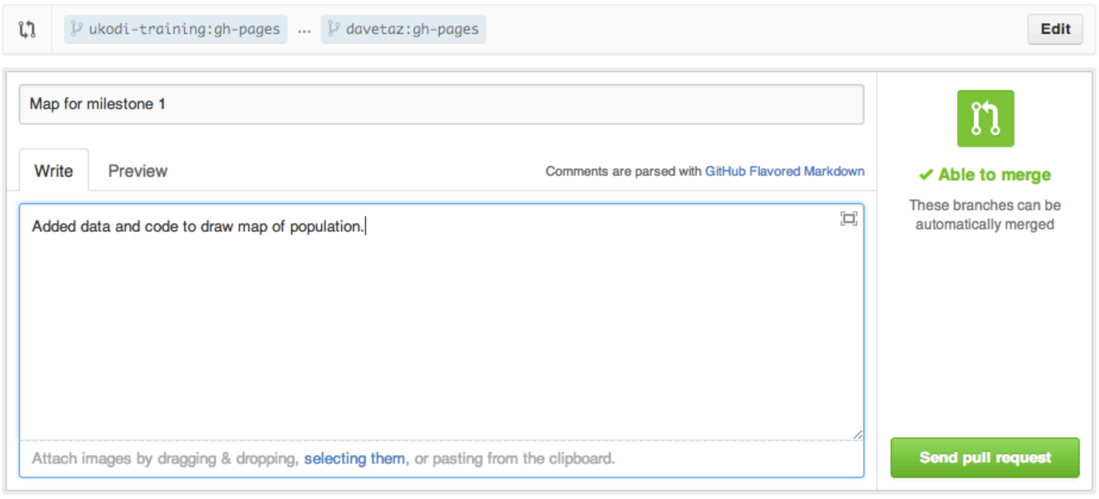
At this point you should have a working map and have closed three issues with your comments. You will also have probably noticed that the issues are not closed in the ukodi-training repository. This is because you are still working on your own repository. In order to close the issues (and related milestone) your changes have to be merged back into the ukodi-training repository. This is done with a **pull request.**

In order to do this, first ensure that you are still editing the **gh-pages** branch of your own repository. Once you are sure you are in the correct location you need to press the **pull request** button located near the top of the screen.



This branch is 1 commit ahead and 0 commits behind master

On the next screen you may be shown the changes that you have made. You should check these carefully before clicking the button at the top to create a pull request.


ukodi-training:gh-pages ... davetaz:gh-pages

Map for milestone 1

Added data and code to draw map of population.

Edit
Write
Preview
Comments are parsed with GitHub Flavored Markdown
✔ Able to merge
These branches can be automatically merged
Attach images by dragging & dropping, selecting them, or pasting from the clipboard.
Send pull request

On the pull request screen you should first check that you are submitting your request to the **ukodi-training:gh-pages** branch from your own **gh-pages** branch (as shown top left). On the right hand side of the screen it will tell you if this request can be merged. This is important as if it can't be merged you may have missed some changes that have been made to the ukodi-training repository, for example someone else might have changed the same file as you (or fixed the bug quicker!).

With both of these aspects checked, fill in the title and description and click **send pull request**.

You should now alert your trainer that there is a pull request and get them to demonstrate what happens when a pull request is received.

Step 9 – Adding interaction (advanced)

In order to add interaction to our map, a number of changes are required to the code that renders the map. This code is contained in the **js/map.js** file. The map is rendered using the D3js library, a complex but very powerful tool for building interactive data visualisations. In map.js it is possible to see a number of different sections that load configuration and then parse over the data in order to draw a map. For example you might want to look at the possible map projections and experiment with alternatives that only require a one-word change.

At the bottom of the map.js file you will see a number of sections that define what happens when various mouse events occur. It is in these sections that we define our interaction.

In order to make it clear which country a person is hovering their mouse over, we are going to make that countries border thicker. This requires that the following code be added to the **mousemove** function:

```
d3.select(this).style("stroke","black")
               .style("stroke-width","1.2px");
```

Feel free to change the color and width of the line to suit. Don't forget to reset the style back to default when the mouse leaves a country (**mouseout**). This is basically the same code but with stroke colour "**#666**" and width of "**0.4px**". Completing this step closes the issue related to highlights.

The other issue relates to displaying the country name and population somewhere else on our web page. In order to do this you will need to add a new section called "**info**" to the index.html page (see step 5) and then add the following to the **mousemove** function in map.js:

```
$('#info').html("<h2>" + d.name + "</h2>Population: <span class='population'>" +
d.value.toString().replace(/\B(?=(\d{3})+(?!\d))/g, ",") + "</span>");
```

Don't forget once you are done to close the relevant issues and submit your pull request!