Managing Open Data with GitHub

In this exercise we are going to look at how we can use Git to manage 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 data repository.

For the purposes of this exercise 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:

https://github.com/theodi/training-data

Opening a web browser at this location should show you the main GitHub screen with the "Code" tab highlighted. You will notice that there are two files, a README.md and a CSV file containing a boiler plate for our data. Feel free to click on both of these files to take a look at their contents.

Clicking on the network tab will reveal the changes graph for this repository, while it should be blank at this point, later on it is likely to get a lot busier, so keep it in mind.

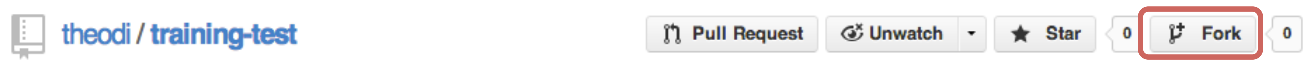
The other tab of note is the Issues tab, here you will note a whole list of issues outlining the lack of data that exists in the data file. While these relate to every group, if you click on the *Milestones* link, you will see that each group has their own milestone containing only the issues relevant for them to fix. You will fix these milestones by adding your own data.

In order to do this, we are going to consider each group a mini-organisation, responsible for completing their own milestone.

Step 2 – Fork the Dataset (Once per Group)

You should designate one person in your group be the group leader. They take a **master** copy for the group following the steps below. We will then repeat this process again with each group member.

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.



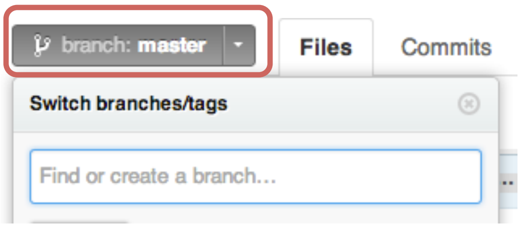
Once you have forked the repository you will be re-directed to a location where you are free to edit the files without danger of affecting the master version.

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 – Branching

Once everyone has access to the repository you have set up, you now have formed a development team. Each member of the team will now need to branch this repository to make their own changes, we are doing this for quality control purposes as **no one should ever commit directly to the master**.

This can be done by clicking the "branch" button. This button will currently be showing that you are on the master branch. Note, your team members could also do this with their local repositories however they gain no clarity from this operation due to the fact they will not be doing merges of the data.



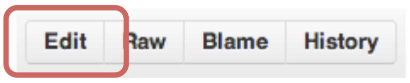
You can choose your own name for the branch, but since you might want to keep roles separate I would recommend choosing GxPy where x is your group number and y is the person in your group (as a number).

Once created clicking the same **branch** button will allow you to switch between your **branch** and the **master** repository. For the next stage you will need your newly created branch to be selected.

Step 4 – Adding Data

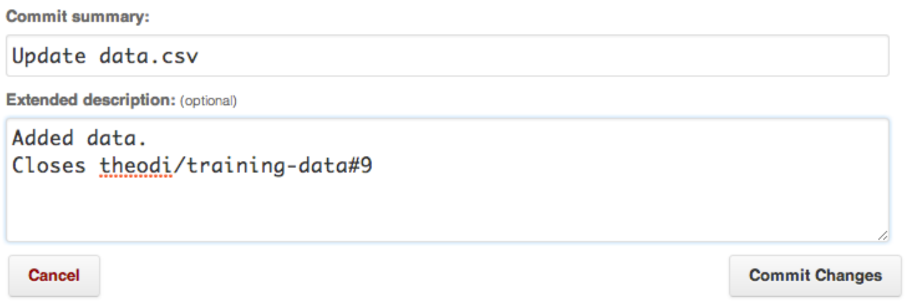
With your repository now showing the files (from the **code** tab), you will want to click on the data csv file to show its contents.

In order to edit this file there is an edit button on the right hand side of the title bar, click this to change the view to an editorial one and then fill in your own data.



IMPORTANT: Only change a single line that corresponds to you, by allotted group number and person number within that group (self allocated).

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 you like (I recommend including your GxPy in the summary) however at some point in the **description** you **MUST** include a link to the issue you are closing in the master "theodi/training-test" 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 #9.



Commit summary:

```
Update data.csv
```

Extended description: (optional)

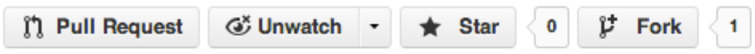
```
Added data.
Closes theodi/training-data#9
```

When done simply click "Commit Changes".

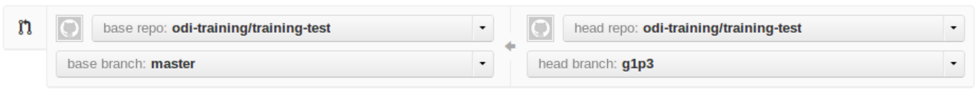
Step 4 – Sending Pull Requests (Part 1)

Each group should now have a repository with four branches, to view this you could look at the **network** tab. In order to merge all your individual changes back into the master we need to create a series of **pull requests**.

In order to do this, first ensure that you are still editing your own branch using the **branch** button. 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.

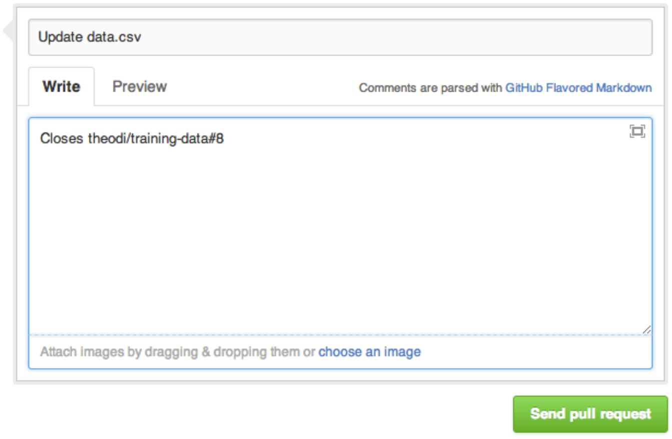


On the next screen you will be presented with a series of options, the first of which you need to check is that you are creating a pull request that submits data from your own branch into the master branch of your teams repository. You should not be referencing anything outside of your local repository at this point.


base repo: odi-training/training-test
base branch: master
head repo: odi-training/training-test
head branch: g1p3

In the example above, our team is odi-training and the repository is training-test. Note that the source is on the right and the destination on the left.

In the box below, you should ensure that your commit messages percolate, especially the one about closing an issue in the master theodi/training-data repository.


Update data.csv

Closes theodi/training-data#8

Write
Preview
Comments are parsed with GitHub Flavored Markdown
Attach images by dragging & dropping them or choose an image
Send pull request

Once done, click **send pull request**.

Step 5 – Merging Changes

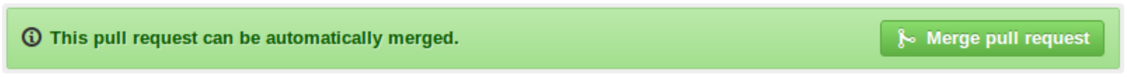
Once each member of the group has submitted a pull request, it is time for you to review your fellow "developers" changes and merge them. All members of the team should now switch to the master branch using the **branch** button.

From here you should now see that the **Pull Requests** tab has **4** outstanding. By clicking the **Pull Requests** tab you can view these 4 requests, each should be from a different member of the team.

Each member of the team should pick one that is not their own to review (I recommend operating a Py$_{+1}$ system). Click on the title of the request to show the detail screen.

On the next screen there are three tabs labeled **discussion, Commits** and **Files Changed**, I recommend you look at all three to see what your team member did.

With luck, on the **discussion** screen you will see this banner:



If you don't then your team member might have touched another line in the data file, causing confusion. You may have to reject the request and ask them to try again (you can delete branches from the **branches** tab).

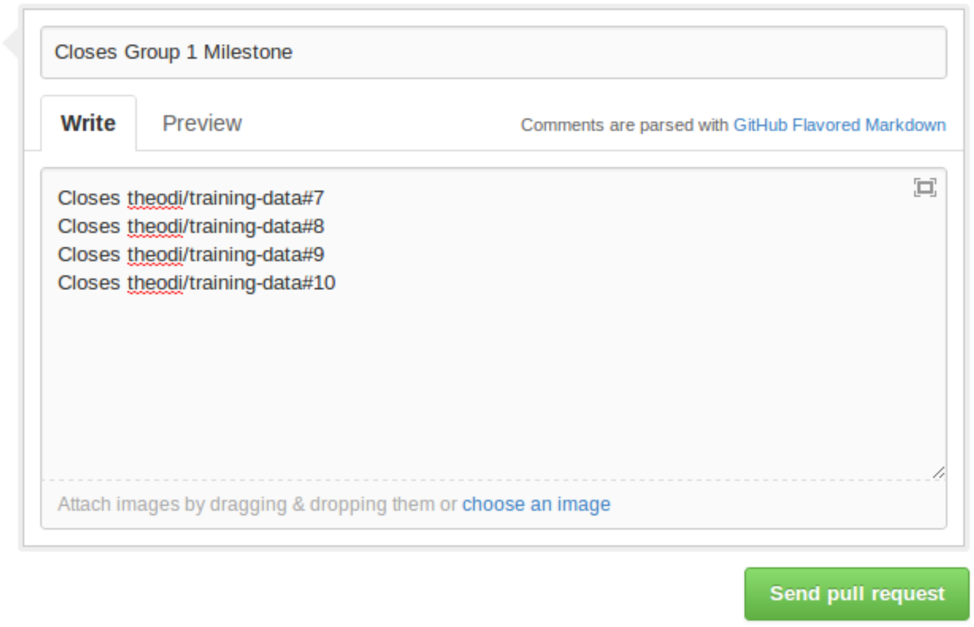
When you merge the branch, ensure you carry forward the "closes theodi/training-data#..." message from the persons commit.

Step 6 – Sending Pull Requests (Part 2)

You should only proceed at this point once **all** of the branches have been merged as we are now going to send a pull request to the main data repository "theodi/training-data". There should be **one per group.**

To send a pull request, repeat the steps from **step 4** this time ensuring that your source is your local master and the target is the remote master of "theodi/training-data", the one you originally forked your repository from.

In the comments section provide a summary that details which milestone you are closing. In the details you should summarize (again) what issues this pull request closes, this time it should be all of the ones relevant to your group (if you have closed them all).


Write
Preview
Comments are parsed with GitHub Flavored Markdown
Attach images by dragging & dropping them or choose an image
Send pull request

Closes Group 1 Milestone

```
Closes theodi/training-data#7
Closes theodi/training-data#8
Closes theodi/training-data#9
Closes theodi/training-data#10
```

Step 7 – Merging Changes

This is essentially the same as **step 5** except that only the trainer who owns the upstream repository you all forked can do this. If they haven't already noticed your pull request, then alert them that it is ready and hope that you pass the data quality control process and that your request can be merged.

Congratulations, you just submitted a data patch. Remember though, if we find bad data, we can track the changes to see who to blame and hit the **blame** button.

As a final step you might want to start exploring all the graphs and histories of both your own repository and the original one that all the data should now be in.