**GitHub User Guide**

**What is GitHub?**

GitHub is an online platform for collaborative projects that uses the Git version control system. GitHub is similiar to cloud-based platforms such as Dropbox and Box in that files exist in the cloud, but users download and edit files on their local computers. In this way, files can easily be shared across users. GitHub is distinct in the way in which it tracks and stores changes to files. GitHub creates a new version of a file only when a user 'commits' a change, and writes an accompanying descriptive message. GitHub also allows easy comparison of changes across different versions, and allows users to revert back to them. Finally, GitHub offers a well developed platform for commenting on code and tracking various issues that arise in collaborating on files. GitHub is used by a wide variety of organizations, hosts a large number of open-sourced projects and also has a very active help community.

**Installing Software**

* Sign up for GitHub [here](https://github.com/signup/free), creating a username and password
* Download GitHub for Windows. GitHub for Windows is a GUI client that enables you to work on files off-line before making them public: <http://windows.github.com/>
* Enter in the username/password that you used for GitHub itself.



**Setting up a GitHub Repository**

Log in to your GitHub.com account:

Once logged in you will be brought to a front page (dashboard) where you can get started on creating your own repository. A repository is the space, essentially a folder, in which a project lives — i.e. it is the central location in which your files can be found and manipulated. On the dashboard you'll see your username in the upper left hand side of the page, a news feed that describes your recent activity, and a list of your repositories on the lower right.

To create a new repository, click on the green "New repository" button or the icon next to your username at the upper-right hand side of the page.

You can now create a name for your repository and write a short description. The default option is that your repository is publicly available to be accessed, while working with private repositories requires a paid GitHub account. You have the option to make a ReadMe file describing your repository (recommended), have GitHub ignore certain types of files, and add a license. When done, click on "Create repository."

**Using GitHub**

**Uploading Files**

The first step is to get your files onto GitHub. We'll describe here the approach with GitHub for Windows, though Git (on the command line) can also be used.

In GitHub for Windows, under the "GitHub" tab, you'll see all repositories you've created online. Clicking on the repository name clones the (currently empty) repository onto your local computer. GitHub has created a folder on your computer (documents/GitHub) into which you can add files into particular repositories (clicking on "open in explorer" will send you directly to the folder). Once a file has been added, it is ready to be "committed" (i.e. saved) to the repository. Checking the "local" tab (right above the "Github" tab) will show your local repositories and any uncommitted files. Below a simple text file was created and added to a folder called "test1."

After typing in a commit message and an optional (but recommended) description, the file can be committed and synced to GitHub.com. Click "publish" at the top of the screen to do this. You should now see the "publish" button replaced with "in sync" and "no uncommitted changes" on the right side, as shown below.

Returning to GitHub.com, your file will now be in your repository. Repeat the process for all files you wish to add.

**Editing Files**

Once your files are up on GitHub.com there are several options for using and editing them. From your main repository page, clicking on your file allows you to open it in GitHub for Windows or directly in the browser. Clicking "Edit" allows you to edit the file directly and then commit those changes back into the main file. Note that for certain types of files (e.g. .dta) you cannot view in the browser and would have to download them before making changes.

If you click the "History" button, you'll see a record of the commits you've made. You can see your code at various points in time by clicking "Browse code" at the bottom right of each entry. Clicking the "Blame" button is very useful feature that allows you to see all changes that were made to your file. See below for an example:

Clicking on the random series of letters and numbers associated with each change (the 'commit sha') shows a "diff" of the changes made at that point in time. See below:

Lines colored in red reflect deletions to the file while lines in green reflect additions. In the above example, line 5 was changed from "Test change2" to "Test change3" while the rest of the file remained as it was. You can make comments on particular lines of the code (put your cursor just to the left of the line) or on all of the changes in the box at the bottom.

Thus far we have been working on the "master" branch of our repository, but we can also create a new branch, for instance if we want to only to work on some subset of our files. At a later point, we can merge this branch into our master branch. We can create this branch from any point in time. Say we want to create a new branch from the point where the last line of our file was "Test change2." First click the "tree" button at the upper-left of the page. See below:

We then name our branch "test-branch." Now that we have created a new branch, we can work independently on it without changing the original file. This is especially useful when multiple users are working on a particular section of a given file or set of files, as you can divide each user's work into separate branches, avoiding the risk of users working over one another. Merging files is done by implementing a "pull request." Imagine we have now added a new line to the code in the "test-branch," and want to merge it into the master branch. Navigating back to the main repository page, ensuring that the branch selected is "test-branch," we click the green "Compare & pull request" button, as below:

Submitting a pull request sends a message to the master branch requesting that whatever changes we have made be pulled into it. Such merges can be rather simple (e.g. appending a new line to an existing file) or rather complex, (e.g. resolving changes to the same line of code from multiple branches) which would require the use of Git on the command line. As all that was done below was to add one line ("Test merge") to the code, you can see on the right that we are able to merge this back into the master branch directly in GitHub. Now, after making any necessary comments, you can click "Send pull request."

Once the pull request has been submitted, one can review all the changes that have been made, assign particular users to review certain aspects of it, and have a general discussion between multiple users. Users can also make additional commits within a given pull request. This process is simple, given any new commit made to the branch to be merged is automatically added to the pull request. Once all commits have been made and changes reviewed, clicking "Merge pull request" and then "Confirm merge" is the final step in the process — changes will now be reflected in the master branch. For more on pull requests, see [here](https://help.github.com/articles/using-pull-requests).

You can also use branching to revert back to old versions of a file. Recall under the "Blame" tab of a given file we can see all changes that have been made to a file and can view that file at that point in time. Creating a new branch from that point essentially allows us to work off of a previous version of the file.

**Working with Others**

From the above, you can probably see how GitHub can be a powerful collaborative tool. A key feature to discuss specific to multiple users is "forking," which is a process where a user can copy the entirety of a repository, allowing them to make whatever changes to files without affecting the originals. These changes can then be merged back (and/or with other collaborators' changes) by pulling them into the main files. The first step is to simply click the "Fork" button at the top-right of the screen. This creates the copy of the repository, which can then be downloaded onto your local drive using GitHub for Windows.

Once you've forked a repository, you can create new branches off of the fork, submit pull requests to the original repository, and update your fork with any changes that have been made to the original.

As mentioned above, branching is a key component to working collaboratively. Multiple branches can be created according to which files/portions of code that each team member is working on. Each user can work independently (on their local drive) on these branches, without affecting the master branch, and only merging in changes when necessary. As with all of GitHub, all changes are tracked on all branches, making it relatively easy to see which changes were made by which user.

**Other Keys to GitHub**

**GitHub Help and Training**

GitHub has a very large user community and extensive training materials. They offer free [monthly trainings] (<http://training.github.com/>), with Q&A directly following. The [help](https://help.github.com/) section of the website is also quite extensive. A great way to learn is to simply follow one of the thousands of public repositories on GitHub. You can "follow" a specific user on the site or "watch" a project, allowing you to see updates to the project in your dashboard. IPA has collected a wealth of additional Git and GitHub resources, which you can find [here](https://github.com/PovertyAction/github-training/blob/master/External%20resources.md)

**Issues**

Issues are suggested improvements, tasks or questions related to a repository and/or the files found within it. Issues can be created by anyone (for public repositories), and are moderated by repository collaborators. Each issue contains its own discussion forum, can be labeled and assigned to a user. Issues integrate well into commits and pull requests: they can be referenced and even closed directly when making a commit/accepting a pull request. To get started, click the 'Issues' tab on the right of any repository and then 'New Isssue'.

**Private vs. Public Repositories**

The default option is for a repository to be public when it is created. Outside users can access the original code (e.g. by forking the repository) but could not modify it without making changes and having their pull request accepted by the repository's collaborators. Private repositories, whose code/files cannot be viewed unless a user is expressly invited, are available under all paid GitHub accounts. See [here](https://github.com/pricing) for GitHub's pricing plans.