GitHub

GitHub is a cloud-based hosting service for users to upload and track their work. Usually, this work is code-based. Cloud-based means that GitHub provides on-demand resources to its users over the internet, or the cloud. For example, it offers storage space, so we don't have to keep large files on our computers.

Uses of GitHub

The primary uses for GitHub are storing and keeping track of projects and files and collaborating with others. It also acts like a social network, allowing us to connect with other users. There are also several open-source projects (projects that are open to the public) that anyone can learn from, practice with, or edit.

Git vs GitHub

Git is a version control software and can be used independently from GitHub or another hosting platform. GitHub is a platform that enhances Git to make it easier to manage projects and collaborate. It is entirely dependent on Git. We cannot use GitHub without Git!

.gitignore:

This is like the blocked phone numbers list on the phone. If we don't want someone to call us, we add their number to the block list! The gitignore file blocks or ignores specific files from being committed or saved into our repo. These files usually contain confidential information or are system files.

Issues Section:

The following section is called Issues. Issues are where we track tasks or problems and also communicate with others. We don't have any issues at the moment.

Pull Request:

The next section is called Pull Requests. Pull requests or PR for short, is a request to make a change to the project. Think of it like a suggestion box. A PR will show the suggested changes and compare them to the project's current version. We can view the suggestion and decide if we want to accept it.

Readme File:

README files need to be descriptive to allow anyone to understand our project, the contents of our repository, and how to use it. It is the first file a person will see, and as a data professional, it is a crucial skill to communicate a project to others. Consider it to be the instruction manual for our repository.

Some essential items are the project title, a description of the technology used and why, a description of the process used to answer the question and why, and a table of contents.

Other characteristics of a good README are if it includes details of how the project came about and the motivation behind it, any limitations or challenges encountered, a recap of the problem it intends to solve, and what its intended use is. If we are sourcing information from elsewhere, it’s also essential to include the necessary credits.

Creating a File in Repo:

If we want to create a file, we select the add file button and choose create new file. We enter the name of our file at the top, including a file extension. Here we create requirements-dot-txt, a text file containing the Python packages for our project.

Making changes to a file:

we have two options for how to save the file. We can commit directly into our current branch or create a new branch for this commit and start a pull request.

Branches:

branches are used for concurrent work on different parts of a project. They can also reduce the risk of conflicting versions of files. For example, we may analyze Spanish soccer data in a branch called la liga, while a colleague analyzes English Premier League data in a branch called premier league. When we create our repo, there will be one branch called main. In our repo's Code section, we can see which branch we are currently located in. We click on this icon if we want to create a la liga branch. This lists all branches. Notice main is the Default branch, meaning it was created when we made the repo. We then click the New branch button on the right.

Rules of Merging Branches into Main:

GitHub offers a way to enforce rules for how we use specific branches. For example, we can require a pull request, which we will discuss later, before branches can be merged. This adds a layer of protection against bringing incorrect code into our main branch. We can also set a rule requiring pull requests to be approved before a merge can occur, which may help improve code quality. We might also restrict who can delete a protected branch.

Adding Rules to Branch Merging:

We can add rules to branch merging from settings and then in branches.

Giving Access in Private Repo:

We can give access to collaborators

Personal Access Token:

A PAT is an alternative to a password when authentication is required for terminal commands. As the output showed us earlier, it is required for interacting with remote repos since August 2021. They are used because they are more secure than entering a password. Note that a PAT is not required when using GitHub, only when interacting with a remote repo via the terminal. We need to create a PAT to clone the bank marketing repo, so let's see how to do this.

Cloning Repo:

Cloning a repo is similar to copy-paste; however, it has a link to the original repo. It creates a copy of the repo in its current state on our local computer and allows us to send updates back and forth. We can work on the files locally, push changes back to the repo, or pull any new updates to our local version using Git. Anyone can clone a public repo, but a private repo owner needs to grant access.

Cloning A Repo:

we navigate to the dropdown menu in the green Code button.

The HTTPS tab is already selected. We need to copy the URL provided

by clicking on the copy icon next to it.

Next, we open our terminal window, where we need to use Git to perform the clone. First, we check we are in the correct directory, then type the command git clone followed by the URL we copied from GitHub and hit enter. We may be prompted to enter our GitHub username and personal access token. We hit enter again to initiate the clone.

Forking Repo:

We can copy a repo without linking back to it. In GitHub, this is called forking. A fork of a repo creates an independent copy of it on our GitHub, meaning we can run experiments without the risk of anything reaching the original repo. This option is often used for collaboration. Anyone with a GitHub account can fork a public repo, but the owner of a private repo needs to allow this feature. For public repos, this opens up the opportunity for a lot of collaboration if anyone finds our repo and has an idea. After forking, a contributor can make changes to the project and submit them through a pull request. Forking differs from creating a new branch where we need to be registered as project collaborators.

Difference Between Fork and Clone:

**Directly Cloning Someone Else's Repository**

* **Command**: Use the git clone command with the repository's URL:
* git clone https://github.com/username/repository.git
* **What You Get**:
  + A local copy of the repository on your computer.
  + The clone will be linked to the original repository as its origin remote.

**Limitations of Direct Cloning:**

1. **No Ownership**:
   * The cloned repository belongs to the original author on the remote (e.g., GitHub).
   * You cannot push changes directly to the original repository unless you have write access.
2. **Proposing Changes**:
   * If you don’t have write access and want to contribute, you must still fork the repository.
   * After making changes locally, you can push them to your forked repository and submit a pull request to the original repository.
3. **Best Practice**:
   * Fork the repository first if you intend to propose changes or work independently on a project.
   * Directly cloning is sufficient for read-only purposes, such as studying the code or testing it locally.

**When to Use Each:**

* **Direct Clone**: Use when you only need a local copy for personal exploration, debugging, or testing.
* **Fork + Clone**: Use when you intend to contribute to the project or maintain your own independent version on a platform like GitHub.

GitHub issue:

GitHub issues are messages to track problem fixes, plans, important tasks, and communications for a project.

They live under the Issues tab, where we can see all of our issues for that repo in one place.

Assigning an Issue to Solve and Tagging:

Assigning a user clarifies who should be working on the issue, while tagging a user is more of a communication tool to ensure the right people read the message.

Closing an Issue:

Finally, once everything in the issue has been addressed, we can close the issue. This is done by hitting the Close with comment button. The default is to close as completed, indicating that items in this issue were successfully addressed. The other option is to close as not planned, indicating that a fix was not possible, for example.

Pull Request:

A PR is a way to notify others about changes we would like to make to a branch within a repo. It also allows the repo owner to check the changes before they are added. It is best practice to add changes to a separate branch from the default main branch. This ensures that main only contains finished work. Finally, the aim of a successful PR is to merge two branches.