I.Subtract

1. Introduce Git and Github.

-Overview

-What is Git?

-What is Github?

2. Literature review.

-Basic and advanced commands.

-How to use Git and Github?

-Workflow.

-Practical result.

-Conclusion.

***II.Introduce Git and Github.***

Before going to introduce Git and Github we need to know some concepts:

# Version **control**, also known as **source control**, is the practice of tracking and managing changes to software **code**. Version **control** systems are software tools that help software teams manage changes to **source code** over time.

•The system helps to store all changes to the source code.

•Support multiple people to work at the same time.

•Can detect who make change to the code.

•This can revert all the changes, bring code to the old version. People don’t need to be afraid of loose their code.

***1.Overview***

A quick aside: Git and GitHub are ***not***the same thing. Git is an open-source, version control tool created in 2005 by developers working on the Linux operating system; GitHub is a company founded in 2008 that makes tools which integrate with git. You do not need GitHub to use git, but you cannot use GitHub without using git. There are many other alternatives to GitHub, such as GitLab, BitBucket, and “host-your-own” solutions such as gogs and gittea. All of these are referred to in git-speak as “remotes”, and all are completely optional. You do not need to use a remote to use git, but it will make sharing your code with others easier.

***2. What is Git?***

* Git was born in 2005.
* Git was made by Linus Torvalds, who supports to write Linus kernel.
* All the code and history are stored in Personal Computer.

***3. What is Github?***

* Was born in 2008.
* It is a service that provides a web-based Git source code repository for software development projects.
* The Word’s Largest Git Repository.

***Some important keywords we need to know:***

**Repository**: A repository, encompasses the entire collection of files and folders associated with a project, along with each file’s revision history. The file history appears as snapshots in time called commits, and the commits exist as a linked-list relationship, and can be organized into multiple lines of development called branches. Because Git is a DVCS, repositories are self-contained units and anyone who owns a copy of the repository can access the entire codebase and its history. Using the command line or other ease-of-use interfaces, a git repository also allows for: interaction with the history, cloning, creating branches, committing, merging, comparing changes across versions of code, and more.

Working in repositories keeps development projects organized and protected. Developers are encouraged to fix bugs, or create fresh features, without fear of derailing mainline development efforts. Git facilitates this through the use of topic branches: lightweight pointers to commits in history that can be easily created and deprecated when no longer needed.

Through platforms like GitHub, Git also provides more opportunities for project transparency and collaboration. Public repositories help teams work together to build the best possible final product.

Commit: The **git commit** command is one of the core primary functions of **Git**. Prior use of the **git** add command is required to select the changes that will be staged for the next **commit**. Then **git commit** is used to create a snapshot of the staged changes along a timeline of a **Git** projects history.

Branch: In **Git**, **branches** are a part of your everyday development process. **Git branches** are effectively a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug—no matter how big or how small—you spawn a new **branch** to encapsulate your changes.

***III. How to use Git and Github?***

1. **Commands.**

***Some commonly used basic commands in Git that every developer should know:***

- ***Git init***: initializes a brand new Git repository and begins tracking an existing directory. It adds a hidden subfolder within the existing directory that houses the internal data structure required for version control.

- ***Git clone***: coppy a Remote Repo on Github to Local PC. The clone includes all the project’s files, history, and braches.

- ***Git add***: add changes staging area. Git tracks changes to a developer’s codebase, but it’s necessary to stage and take a snapshot of the changes to include them in the project’s history. This command performs staging, the first part of that two-step process. Any changes that are staged will become a part of the next snapshot and a part of the project’s history. Staging and committing separately gives developers complete control over the history of their project without changing how they code and work.

- ***Git commit***: saves the snapshot to the project history and completes the change-tracking process. In short, a commit functions like taking a photo. Anything that’s been staged with ***git add*** will become a part of the snapshot with ***git commit.***

- ***Git push***: updates the remote repository with any commits made locally a branch.

- ***Git status***: shows the status of changes as untracked, modified, or staged.

***- Git pull***: updates the local line of development with updates from its remote counterpart. Developers use this conmmand if a teammate has made commits to a branch on a remote, and they would like to reflect those changes in their local environment.

Some advanced commands:

-***Git branch***: is divided to many branches. Each branch has it’s own task.

- ***Git checkout***: tells Git which branch or commit to apply your new changes.

-***Git merge***: use to combine a branch with another branch.

**2.How to use Git and GitHub ?**

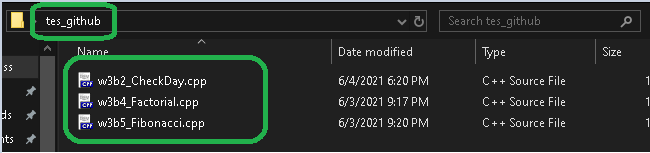
**We have 2 ways to use this program :**

**GUI – Bash**

**Use Bash**

**Step1:** Select the folder (file,project,program,etc..)

For example : I choose this folder name “test\_github”



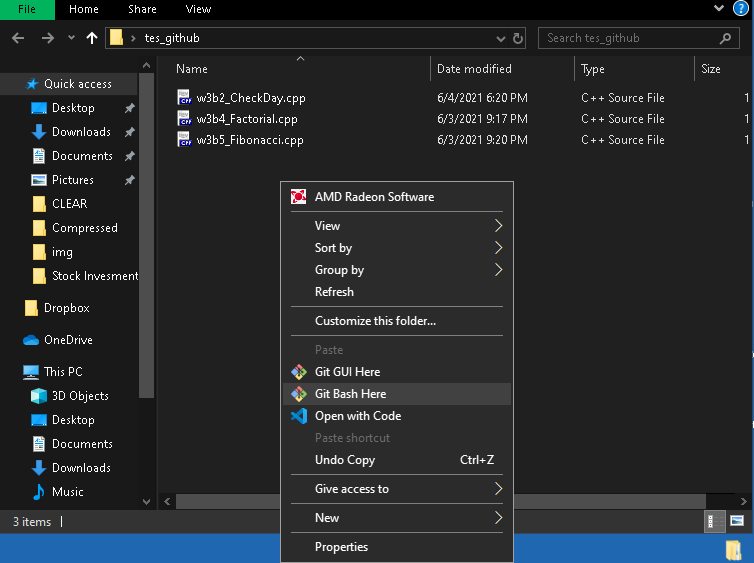
In this folder we have 3 program :

-w3b2\_CheckDay.cpp

-w3b4\_Factorial.cpp

-w3b5\_Fibonacci.cpp

**Step2:** Right click inside folder and go to “Git Bash here”

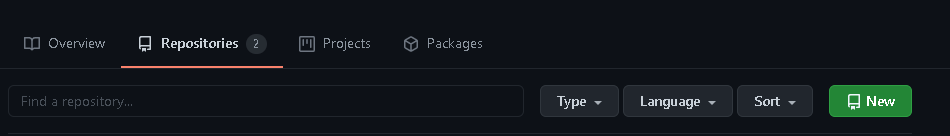


**Step3**: When the terminal opened

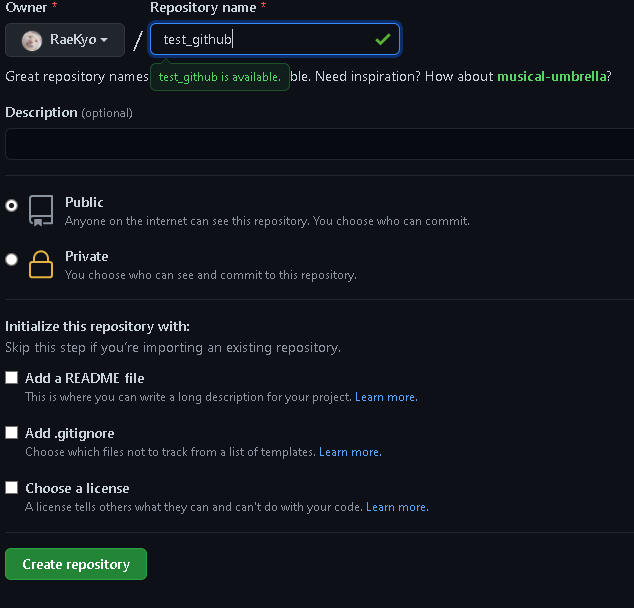


**Step4: Open github.com**

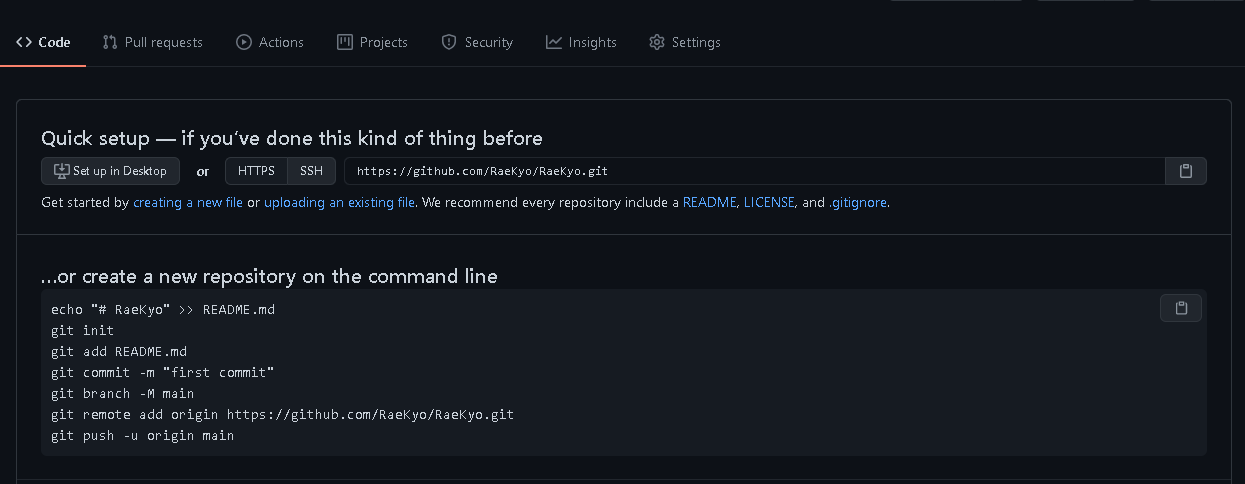
Sign in github.com and click “New” on Repositories



Set the Repository name and click “Create repository”



You will see something like this :



**Step5:**Go back the “terminal windown ”and follow the tutorial to create the repo (repository):

For example : I’ll push all three C(.cpp)

**1)** git init - Create a new repo in this path ~/Desktop/test\_github

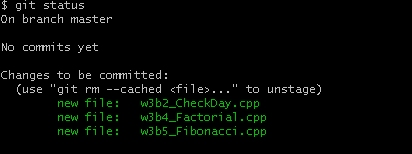


**2)** git add {the program} – to add this file to staging area (git add . : this mean add all file )



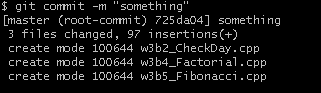
If we want to check what happen to the file we say :

git status



**3)** Three file was on branch master , now go to commit it (when commit something we need to create a message to anyone know what the meaning of the file was pushed) use this flag(-m ) :

git commit -m “something”



Three file was commit complete !!

**4)** git branch -m main , this mean we will create a new branch name “main” but we will push this file directly to ”master branch”🡪 So we can skip it :D

**5)** copy “fully” the sixth command and paste to terminal

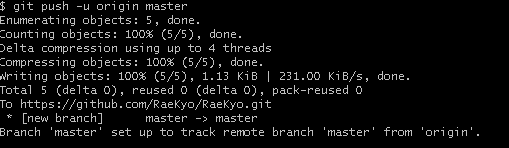


* This mean all command we start with this path will have the URL: [http://github.com/{admin}/{admin.git}](http://github.com/%7badmin%7d/%7badmin.git%7d)

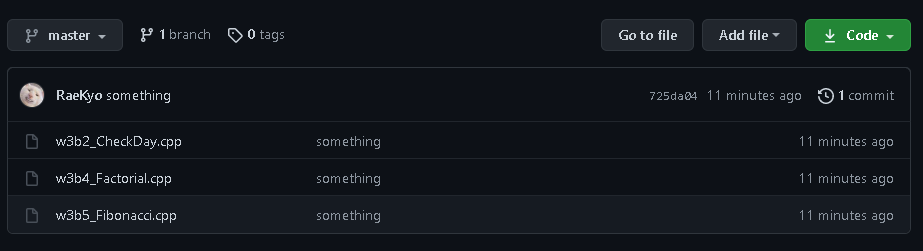
**6)Pushing**

git push -u origin master

we use “master” because I want push it directly to master branch not other one



**Now** , everything are ready 🡪 go back to this website and “Reload” the page or “F5” key



**Using “Git Gui ” is more simple than “Git Bash” : But if you understand how to use command line we can use “git gui easily ”. :D**

**3.Workflow**

A. Workflow with efficiency Github

* To use git effectively in projects, there will be rules and workflows that keep the source code development process in sync, minimizing errors and conflicts. Team members need to follow git working flow rules as follows:
* \* The project's repository must have two branches, master and develop. Where master branch is the stable branch containing the complete code used for production. The develop branch is the branch where the latest source code of the features is developed and is being deployed to the server for testing.
* \*Software developers need to fork the project's original repository into one of their own, when developing a new feature or fixing a bug, the programmer needs to create a new branch from his repository before performing the work. work on that branch. When the work is done, commit your code to the branch then make a merge request to merge your code into the project's original repository.
* \*Need to pull code from the original repository when developing a new feature to ensure that the code on your machine is the latest code from the project, avoiding conflicts (conflict code) after creating the merge request.
* \*For those who have the right to merge code from other branches into the develop branch, it is necessary to have a thorough review, when a conflict occurs, call the people involved in the development of that feature to be able to solve it in the best way, avoiding the situation. case after merge code does not run or runs completely wrong against the programmer's purpose.

B. Centralized Workflow on Github.

* Let's first learn a little about the successful workflow before getting to the heart of the matter.What is a successful Git workflow?
* When evaluating your team's workflow, it's most important to consider your team's culture. You want workflows to enhance your team's efficiency and not burden productivity. Some things to consider when evaluating Git workflows are:
* -Does this workflow with team size?
* Is it easy to undo errors and errors with this workflow?
* -Does this workflow impose any new unnecessary cognitive costs on the team?
* Centralized Workflow is a great Git workflow for teams moving from SVN (SVN is an open source software used to manage and test different versions of source code during software development.)
* -Like SVN, Workflow Volume Central uses a central repository to act as a single point of access for all changes to the project. Instead of the trunk, the default development branch is called master and all changes are committed to this branch. This workflow does not require any branch other than master.
* -Migrating to a distributed version control system can seem like a daunting task, but you don't need to change your existing workflow to take advantage of Git. Your team can develop projects in exactly the same way they do with SVN.

How it works?

Developers start by cloning the central repository. In local copies of the project, they edit the files and commit the changes as with SVN; however, these new commits are stored locally - they are completely isolated from the central repository. This allows developers to delay upstream synchronization until they are at a convenient breakpoint.

**1.Initialize the central repository**

First, you need to create a central repository on the server. If it's a new project, you can initialize an empty repository. If not, you will need to import an existing Git repository. The central repository will help you save your projects and will be the central storage of your git programs.

**2.Storage central storage**

Central repositories are usually created through 3rd party Git hosting services such as Bitbucket Cloud or Bitbucket Server. The initialization of a bare repository discussed above is handled for you by the service. storage. The storage service will then provide an address for the central repository to access from your local repository.

**3.Clone the central repository**

Next, each developer creates a local copy of the entire project. This will help keep the project backed up in case one of the central repositories fails.

**4.Push new commits to central repository**

When the local repository has new changes committed. These changes will need to be leveraged to be shared with other developers in the project. Since your work as well as your project will be shared with everyone and will be developed not only by you, the part New data is pushed to a central repository to ensure the most complete and accurate sharing of recent changes.

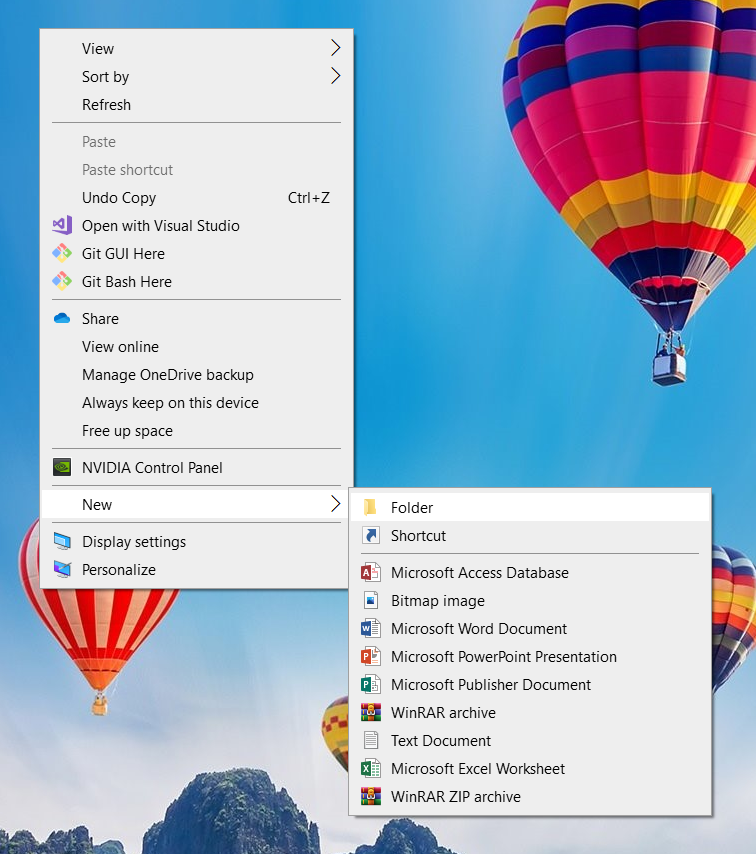
**5.Conflict management**

The central repository represents the official project, so its commit history should be considered sacred and immutable. If a developer's local commits diverged from the central repository, Git would refuse to push their changes as this would overwrite the official commits.

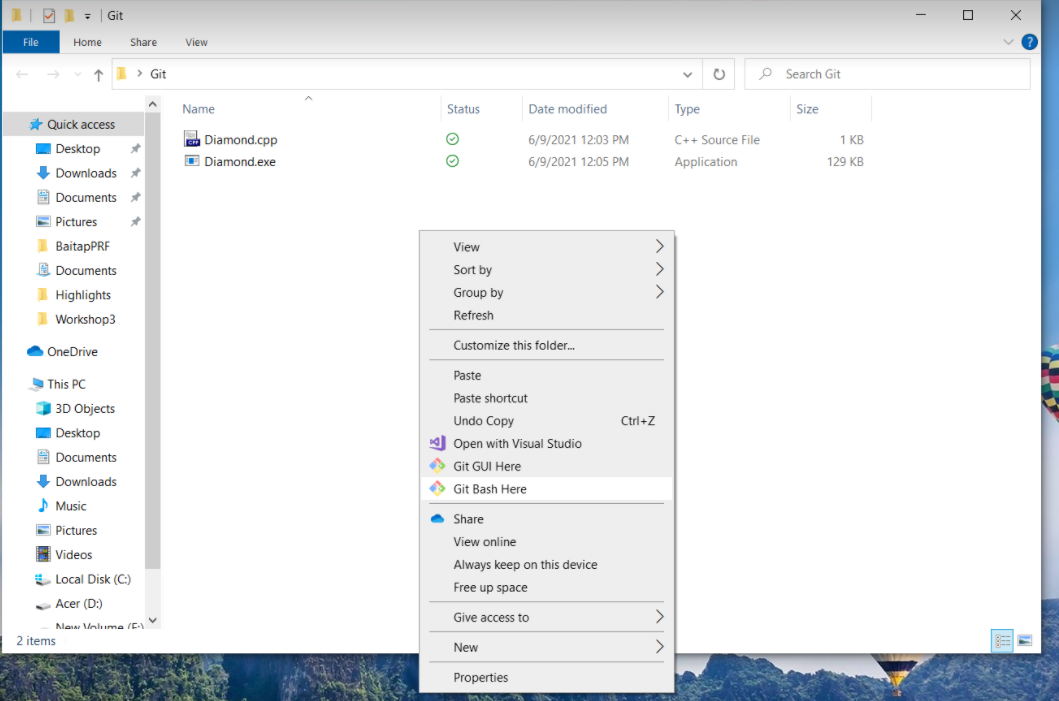
**6.Change and commit**

Once the repository is cloned locally, the developer can make changes using the standard git commit process: edit, stage, and commit. This allows you to create file commits. mid-range, even if you've made a lot of local changes.

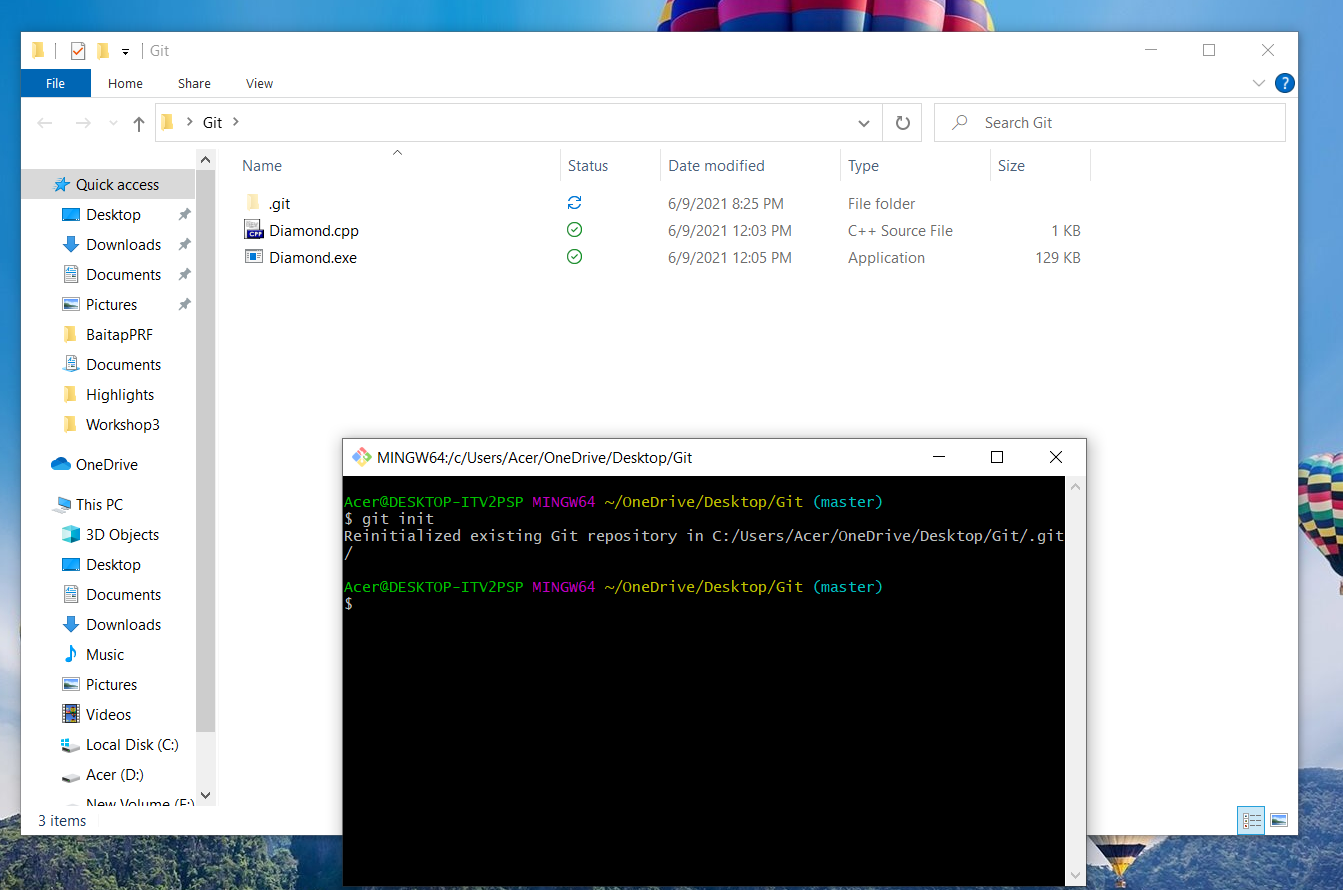
***IV. Practical result***

Step1: Create a folder and let it hold your project 

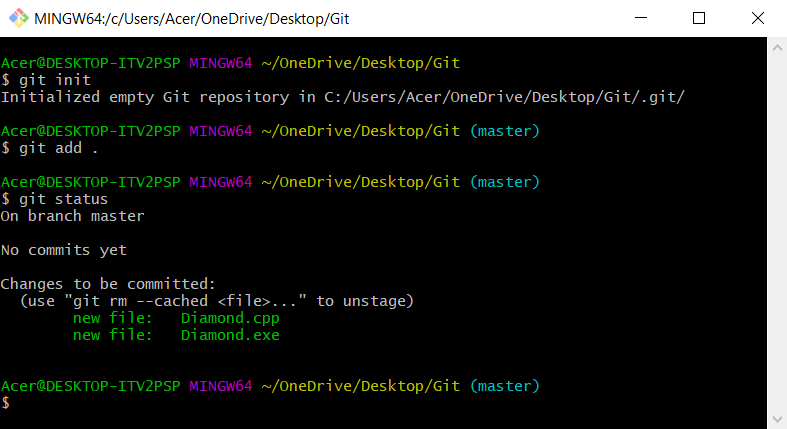
Step 2: Open Git Bash in the folder



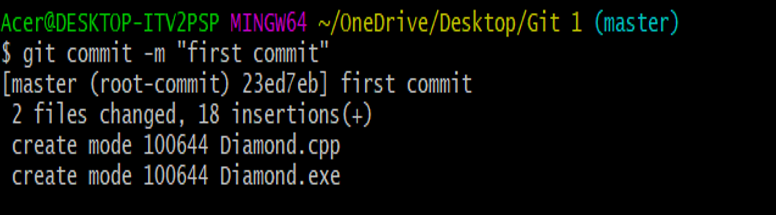
Step 3: Use “git init” to create your repository

****

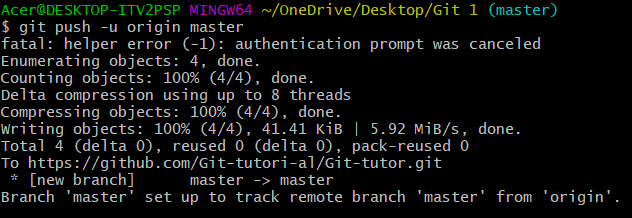
Step 4: Use “git add .” then use “git status” check the stage of repository

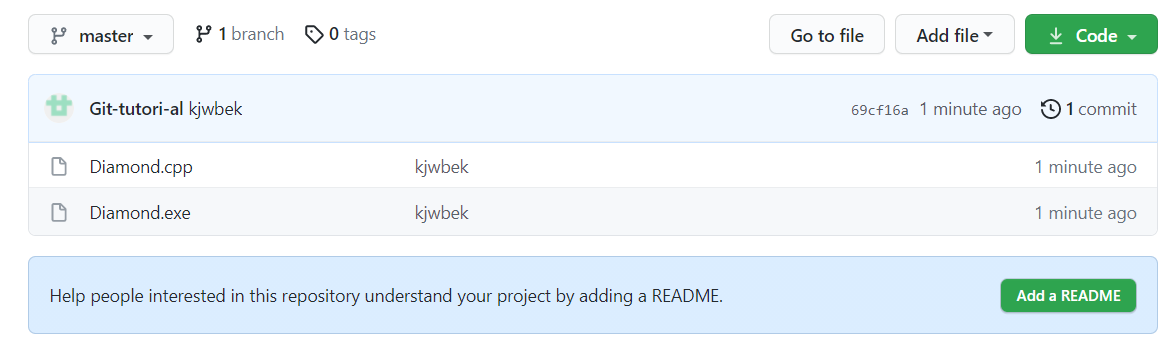


Step 5:Use “git commit –m “message” “ to save file local repo



Step 6: Use “git push –u origin master” to push file in local repo to remote repo.





***V. Conclusion***

Git and Github are the tool that help us work together efficiently and easily and save time and manage our codes more effecienct.

According to the latest Stack Overflow developer survey, more than 70 percent of developers use Git, making it the most-used VCS in the world. Git is commonly used for both open source and commercial software development, **with significant benefits** for individuals, teams and businesses.