Git:

**Git** is an **open-source distributed version control system**. It is designed to handle minor to major projects with high speed and efficiency. It is developed to co-ordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.

What is Git:

What is GitHub:

Difference between git and github:

A sample Work Flow:

Git: Version Control System

Track changes in files/folders

We can collaborate between Teams

|  |
| --- |
| Version Control  Version3  Version 2  Version1 |
| Version 3  Version2  Version1 |

Jithul (File)

Ankit (File)

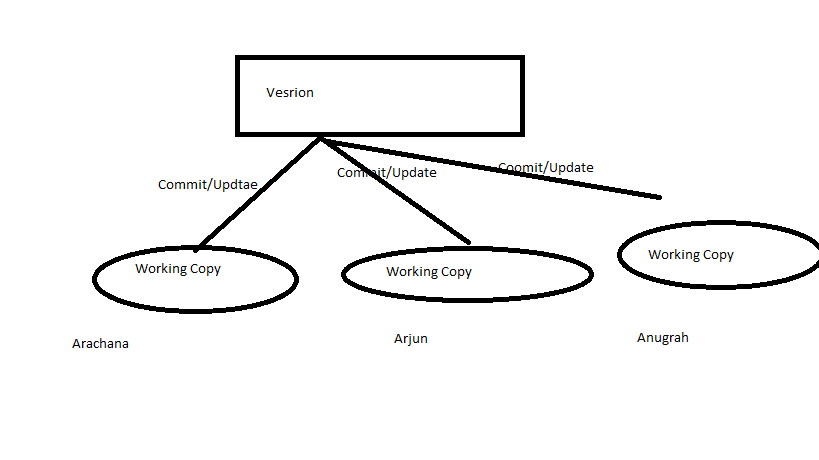
|  |
| --- |
| Checkout  File |

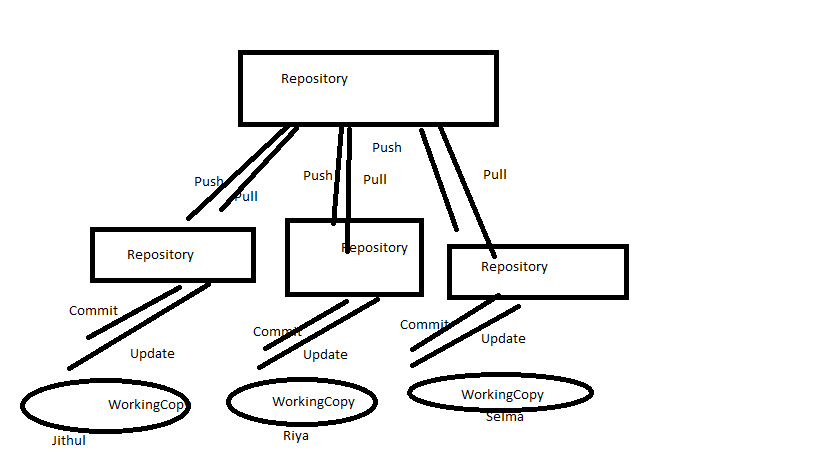
We have Two Types of Version Control Systems:

1)Centralized Version Control System

2)Distributed Version Control System

Centralized Version Control System:

2)Distributed Version Control System:



Git :Distributed Version Control System.

That is the why Git is famous and popular and it is also free and open source.

Git:https://git-scm.com/

GitHub: It is a website to upload your repositories online.

Provides Backup

Provides visual interface to your repo

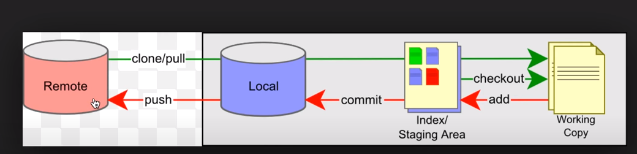
Makes collaboration easier

<https://github.com/>.

Git !=github

Git is a version Control System.

Git hub is a just website to put your repositories



This is a remote repository and this is a local repository Let us say our repository is put on a server its placed on a server which is in this remote server

We can do a clone or we can do a pull to take all the repositories present here into our local system.

So here we can get all the repositories Let us say we have all these four repositories or differenr branches of repositories now we can take or checkout any branch we will create a working copy for us and now we can do all our work on this working copy

And we can be offline during these changes So u can do all the changes and updates here,

Once we are done with our changes we can do our add command and what we will add command to is it will put all ur changes on a staging area it will still not commit it it will just put in staging area and then we have to executecommit commnd, which will make all our changes and committed to the local repository but at this point it is still not pushed it to the remote repository.

So we have to again do push command which will push our changes to the remote repositories

Git is a free, open source distributed version control system tool designed to handle everything from small to very large projects with speed and efficiency.

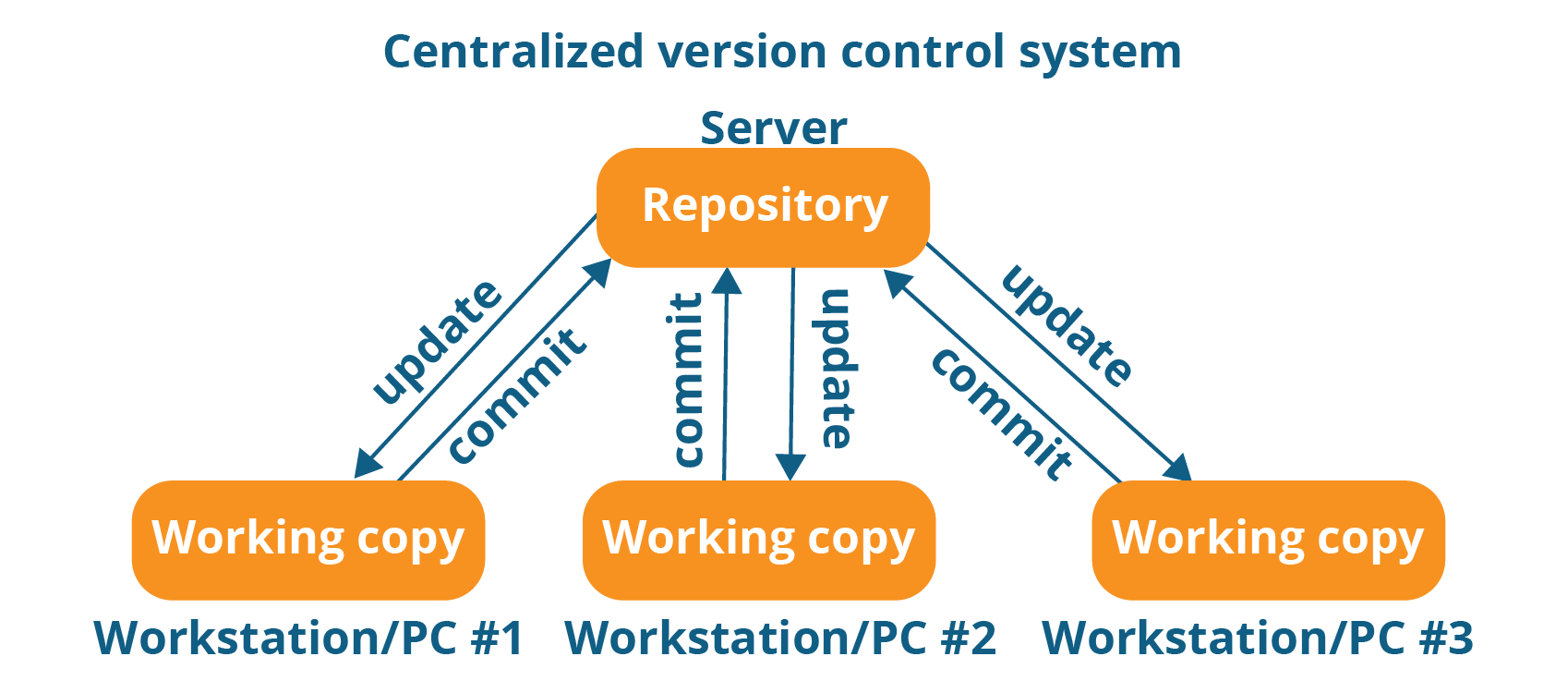
**There are two types of VCS:**

* Centralized Version Control System (CVCS)
* Distributed Version Control System (DVCS)

**Centralized VCS**

Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. It works on a single repository to which users can directly access a central server.

Please refer to the diagram below to get a better idea of CVCS:



The repository in the above diagram indicates a central server that could be local or remote which is directly connected to each of the programmer’s workstation.

Every programmer can extract or **update** their workstations with the data present in the repository or can make changes to the data or **commit** in the repository. Every operation is performed directly on the repository.

Even though it seems pretty convenient to maintain a single repository, it has some major drawbacks. Some of them are:

* It is not locally available; meaning you always need to be connected to a network to perform any action.
* Since everything is centralized, in any case of the central server getting crashed or corrupted will result in losing the entire data of the project.

**Distributed VCS**

These systems do not necessarily rely on a central server to store all the versions of a project file.

In Distributed VCS, every contributor has a local copy or “clone” of the main repository i.e. everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.

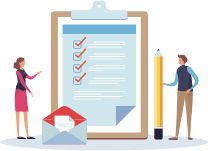
You will understand it better by referring to the diagram below:



As you can see in the above diagram, every programmer maintains a local repository on its own, which is actually the copy or clone of the central repository on their hard drive. They can commit and update their local repository without any interference.

They can update their local repositories with new data from the central server by an operation called “**pull**” and affect changes to the main repository by an operation called “**push**” from their local repository.

The act of cloning an entire repository into your workstation to get a local repository gives you the following advantages:

[[](https://www.edureka.co/devops-certification-training)](https://www.edureka.co/devops-certification-training" \t "_blank)

**[DevOps Certification Training Course](https://www.edureka.co/devops-certification-training" \t "_blank)**

[Explore Curriculum](https://www.edureka.co/devops-certification-training" \t "_blank)

* All operations (except push & pull) are very fast because the tool only needs to access the hard drive, not a remote server. Hence, you do not always need an internet connection.
* Committing new change-sets can be done locally without manipulating the data on the main repository. Once you have a group of change-sets ready, you can push them all at once.
* Since every contributor has a full copy of the project repository, they can share changes with one another if they want to get some feedback before affecting changes in the main repository.
* If the central server gets crashed at any point of time, the lost data can be easily recovered from any one of the contributor’s local repositories.

# Install Git – Git Installation On Windows And CentOS:

## ****Install Git On Windows****

**Step 1**:

To download the latest version of Git, click on the link below:

[***Download Git for Windows***](https://git-scm.com/download/win/)

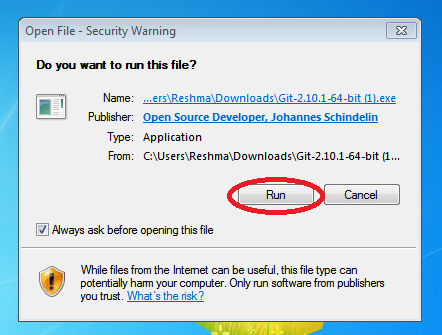
[Git - Downloading Package (git-scm.com)](https://git-scm.com/download/win/)

https://git-scm.com/downloads

Great! Your file is being downloaded.

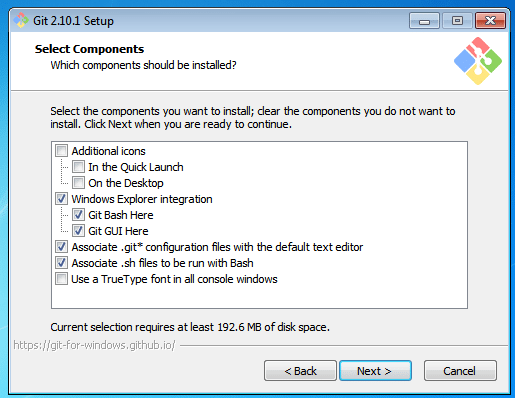
**Step 2:**

After your download is complete, **Run** the .exe file in your system.



**Step 3:**

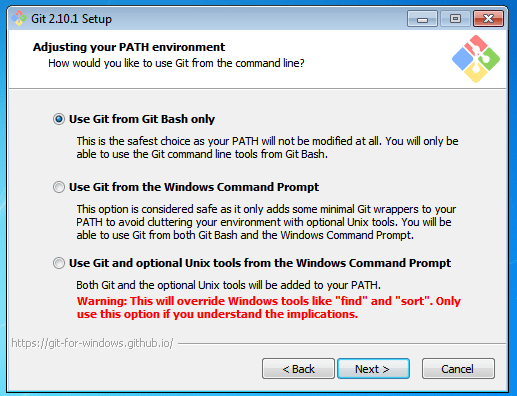
After you have pressed the **Run** button and agreed to the license, you will find a window prompt to select components to be installed.



After you have made selection of your desired components, click on **Next>**.

**Step 4:**

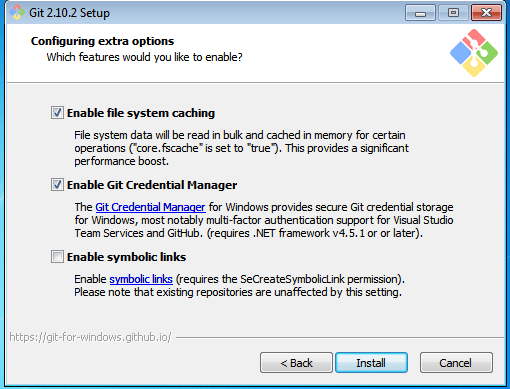
The next prompt window will let you choose the adjustment of your path environment. This is where you decide how do you want to use Git.



You can select any of the three options according to your needs. But for beginners, I recommend using **Use Git From Git Bash Only**

**Step 5:**

The next step is to choose features for your Git. You get three options and you can choose any of them, all of them or none of them as per your needs. Let me tell you what these features are:



The first is the option to **Enable file system caching**.

Caching is enabled through Cache manager, which operates continuously while Windows is running. File data in the system file cache is written to the disk at intervals determined by the operating system, and the memory previously used by that file data is freed.

The second option is to enable **Git Credential Manager**.

The **Git Credential Manager** for Windows (GCM) is a credential helper for Git. It securely stores your credentials in the Windows CM so that you only need to enter them once for each remote repository you access. All future Git commands will reuse the existing credentials.

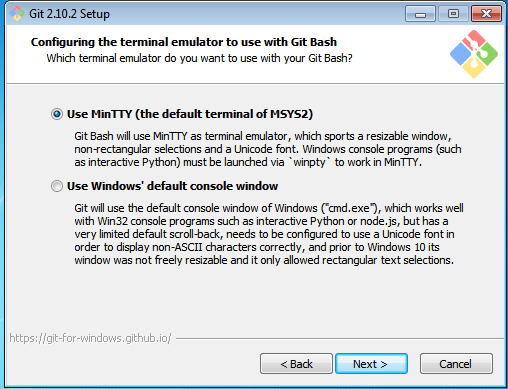
The third option is to **Enable symbolic links**.

Symbolic links or symlinks are nothing but advanced shortcuts. You can create symbolic links for each individual file or folder, and these will appear like they are stored in the folder with symbolic link.

I have selected the first two features only.

**Step 6:**

Choose your terminal.



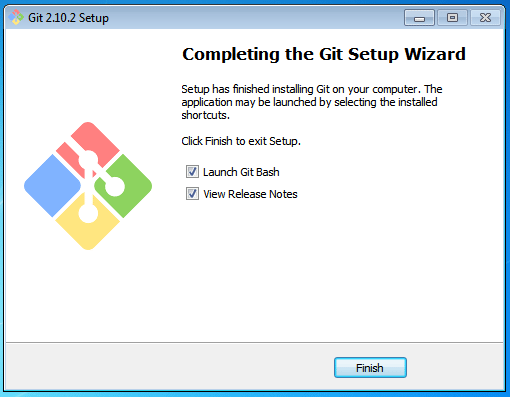
You can choose one from the options.

The default terminal of MYSYS2 which is a collection of GNU utilities like bash, make, gawk and grep to allow building of applications and programs which depend on traditionally UNIX tools to be present.

Or you can choose the window’s default console window (cmd.exe).

**Step 7:**

Now you have got all you need. Select **Launch Git Bash** and click on **Finish**.

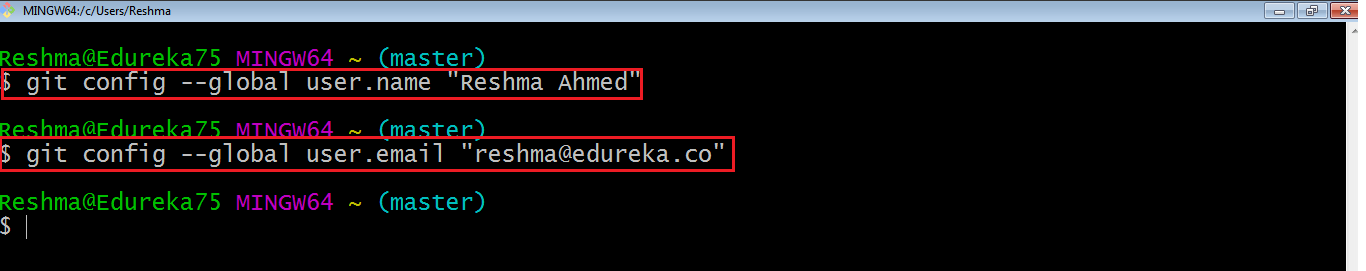


**Step 8:**

Let us proceed with configuring Git with your username and email. In order to do that, type the following commands in your Git Bash:

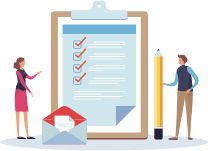
git config - - global user.name "<your name>"

git config - - global user.email "<your email>"



It is important to configure your Git because any commits that you make are associated with your configuration details.

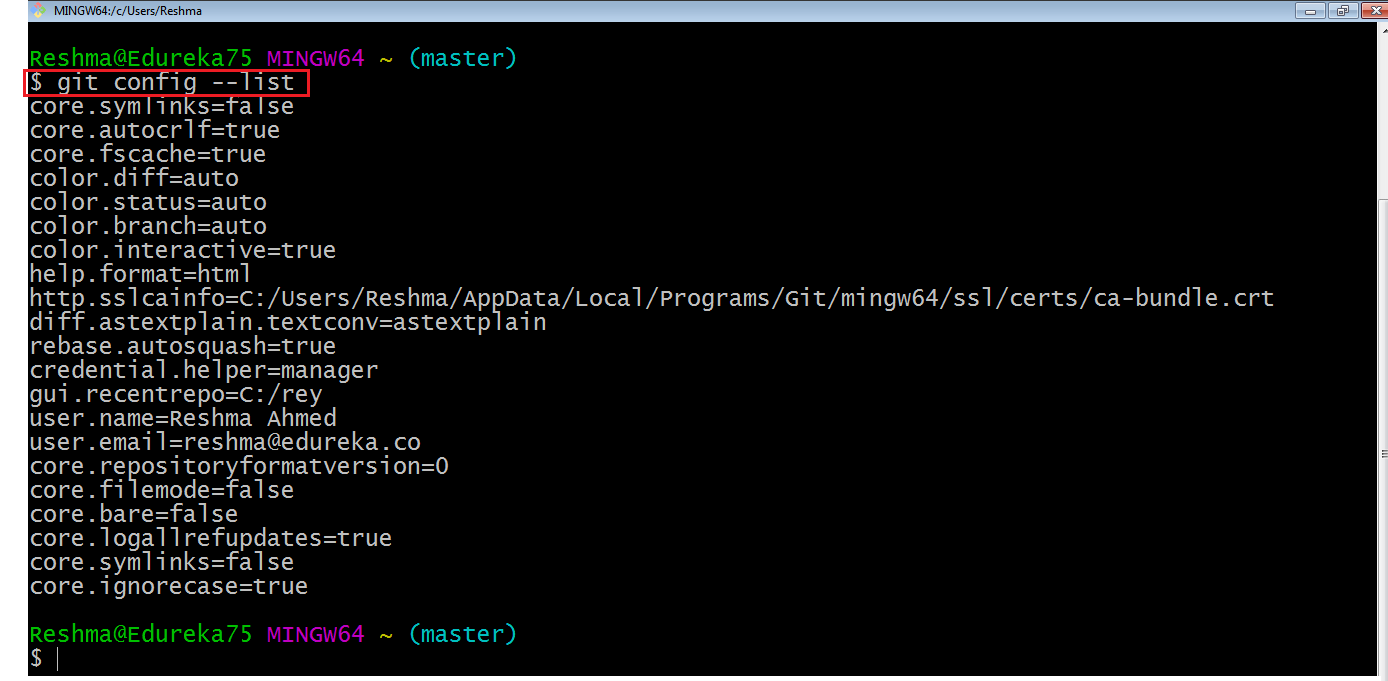
If you want to view all your configuration details, use the command below:

[[](https://www.edureka.co/devops-engineer-training)](https://www.edureka.co/devops-engineer-training" \t "_blank)

### [DevOps Engineer Masters Program](https://www.edureka.co/devops-engineer-training" \t "_blank)

[Explore Curriculum](https://www.edureka.co/devops-engineer-training" \t "_blank)

git config - - list



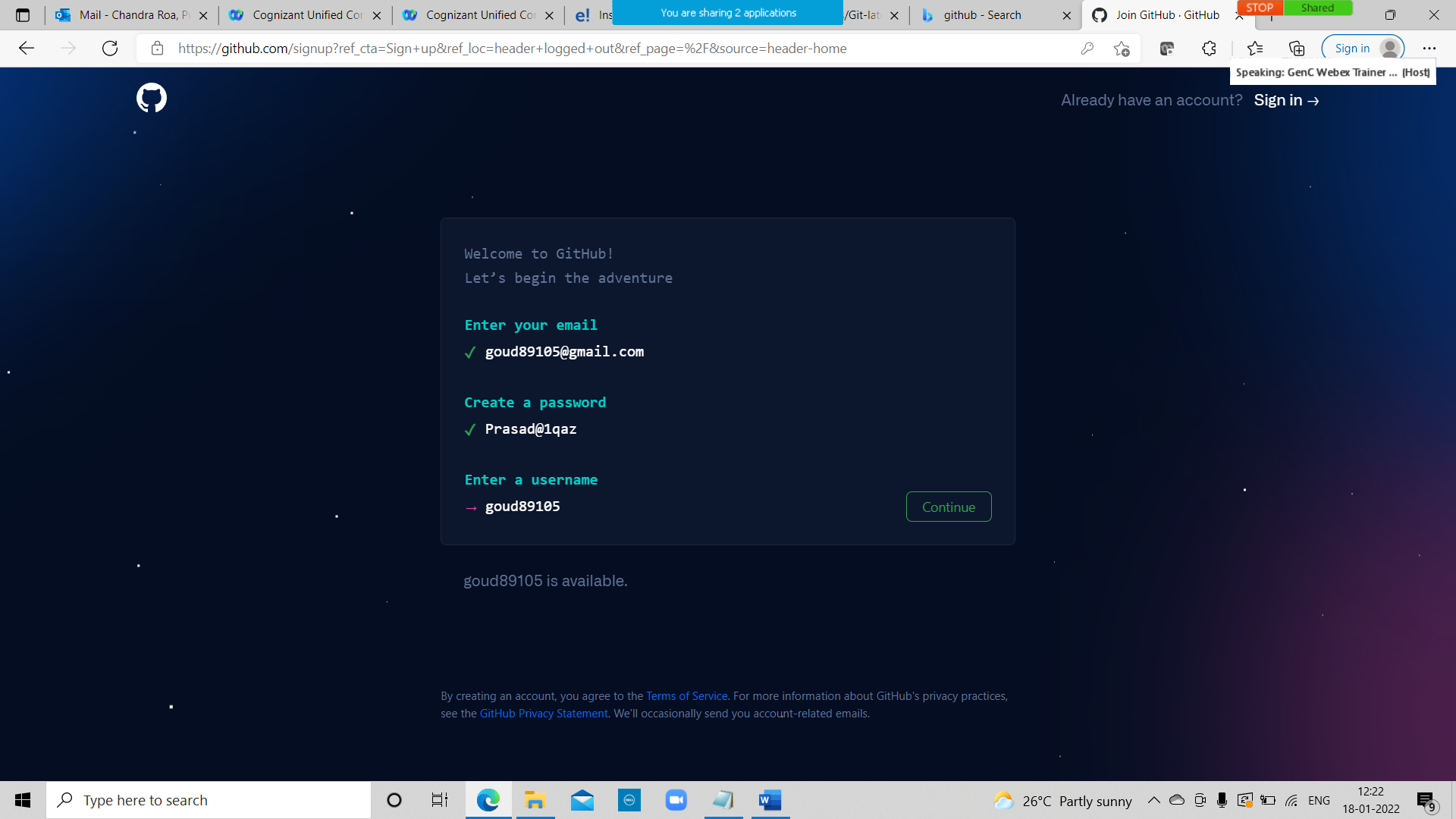
This is how you install and setup GIT on Windows.

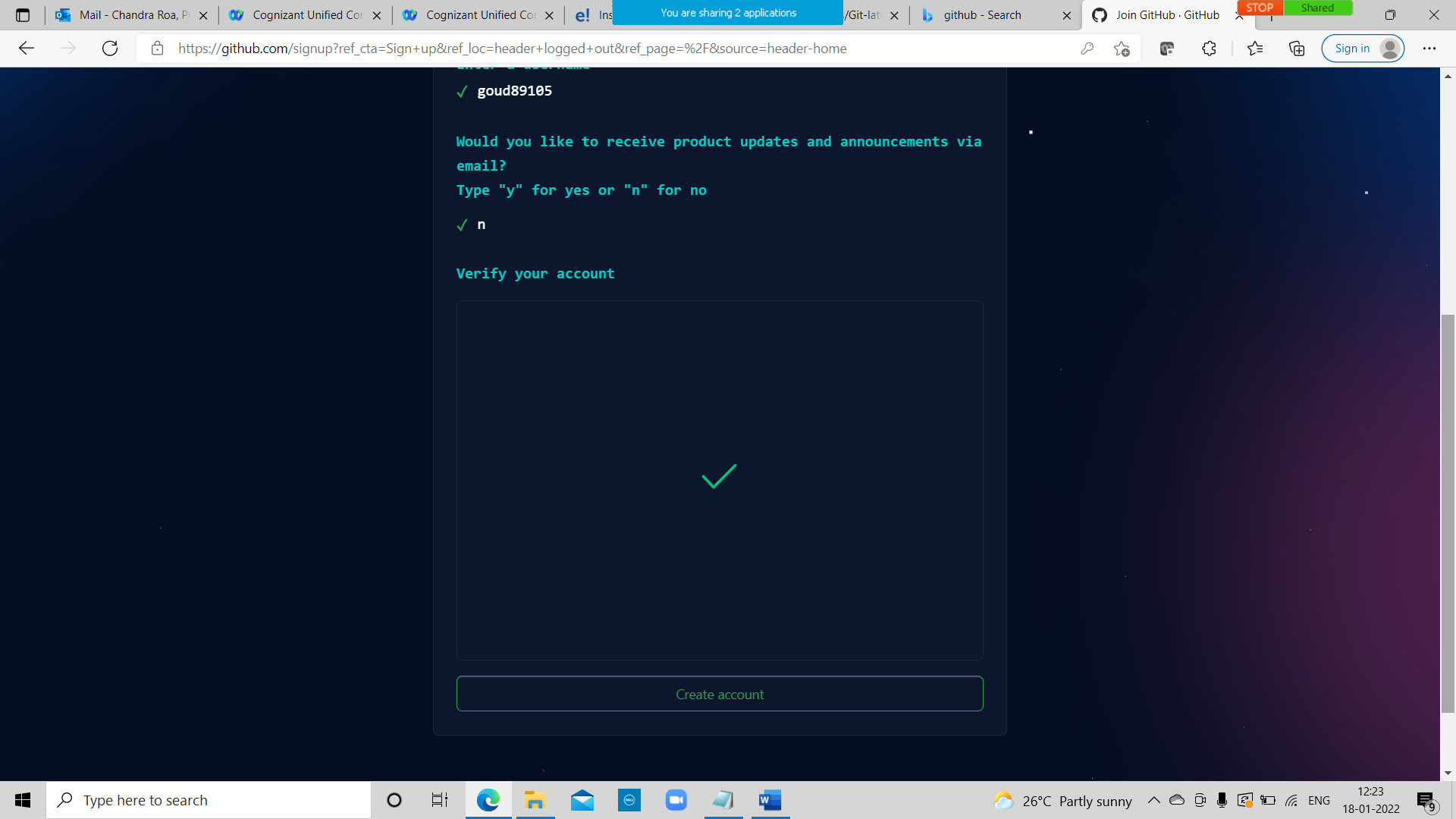
## ****Step 1: What is Github?****

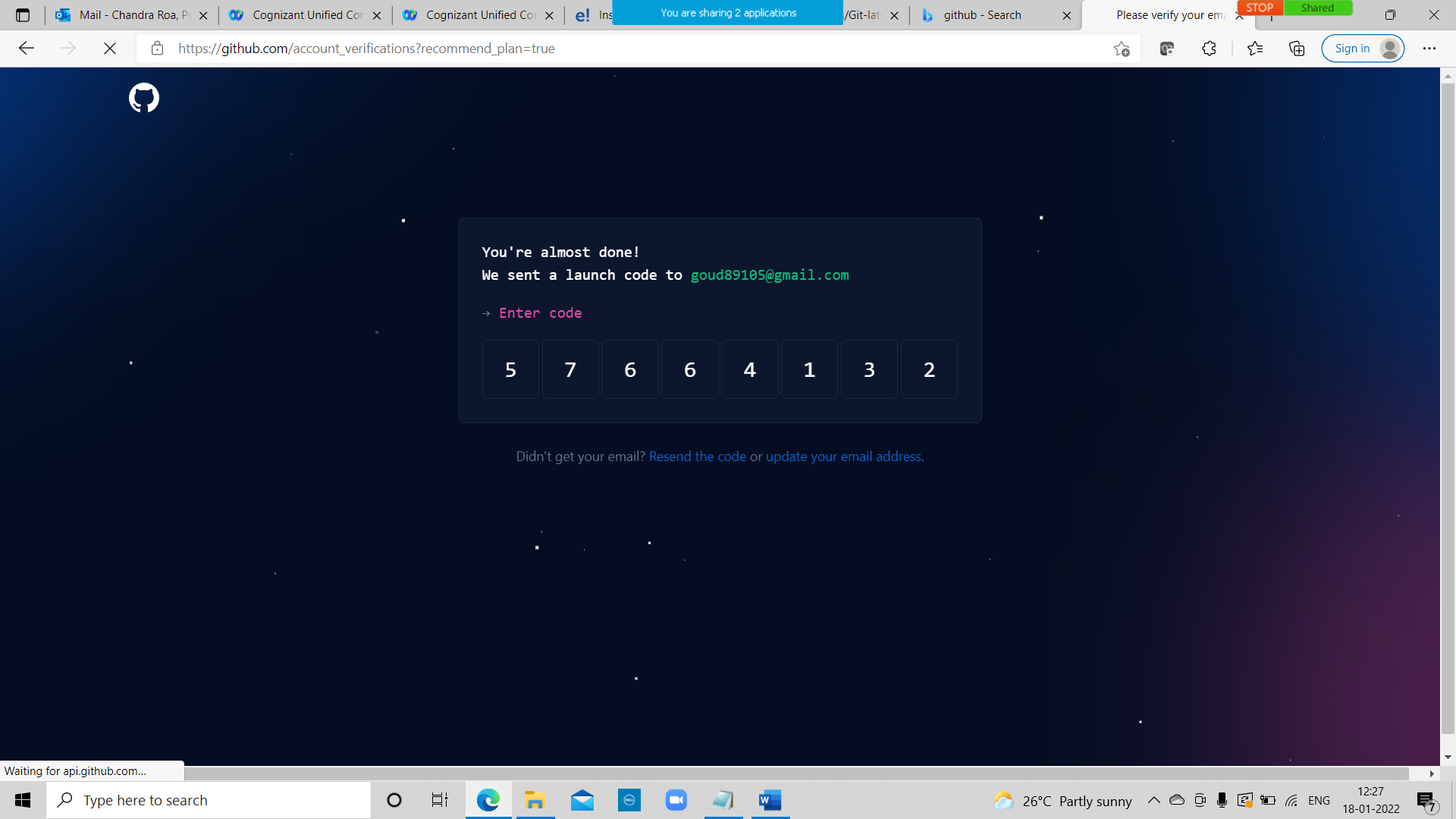
## ****Steps for Git Hub:****

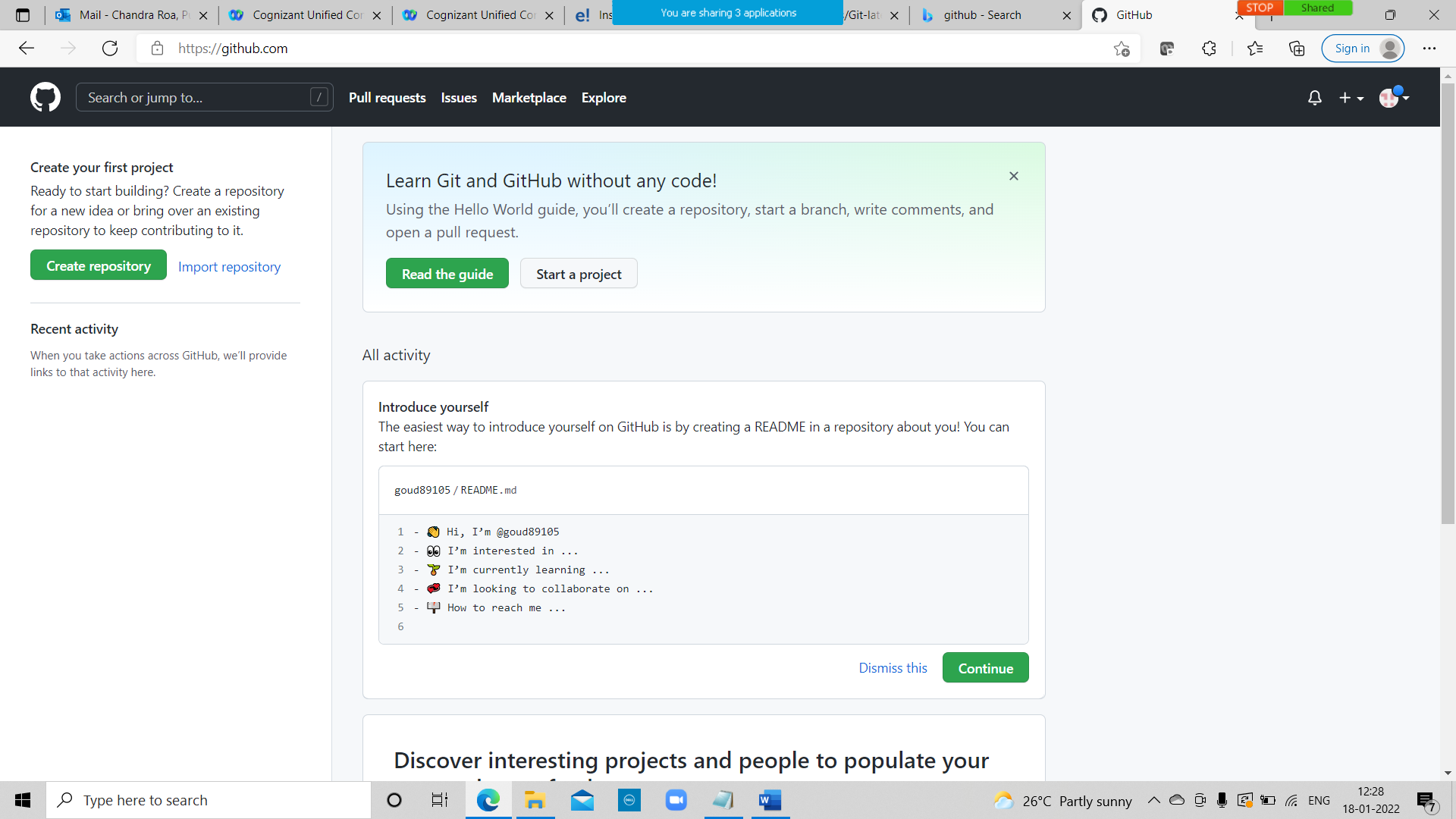
GitHub:

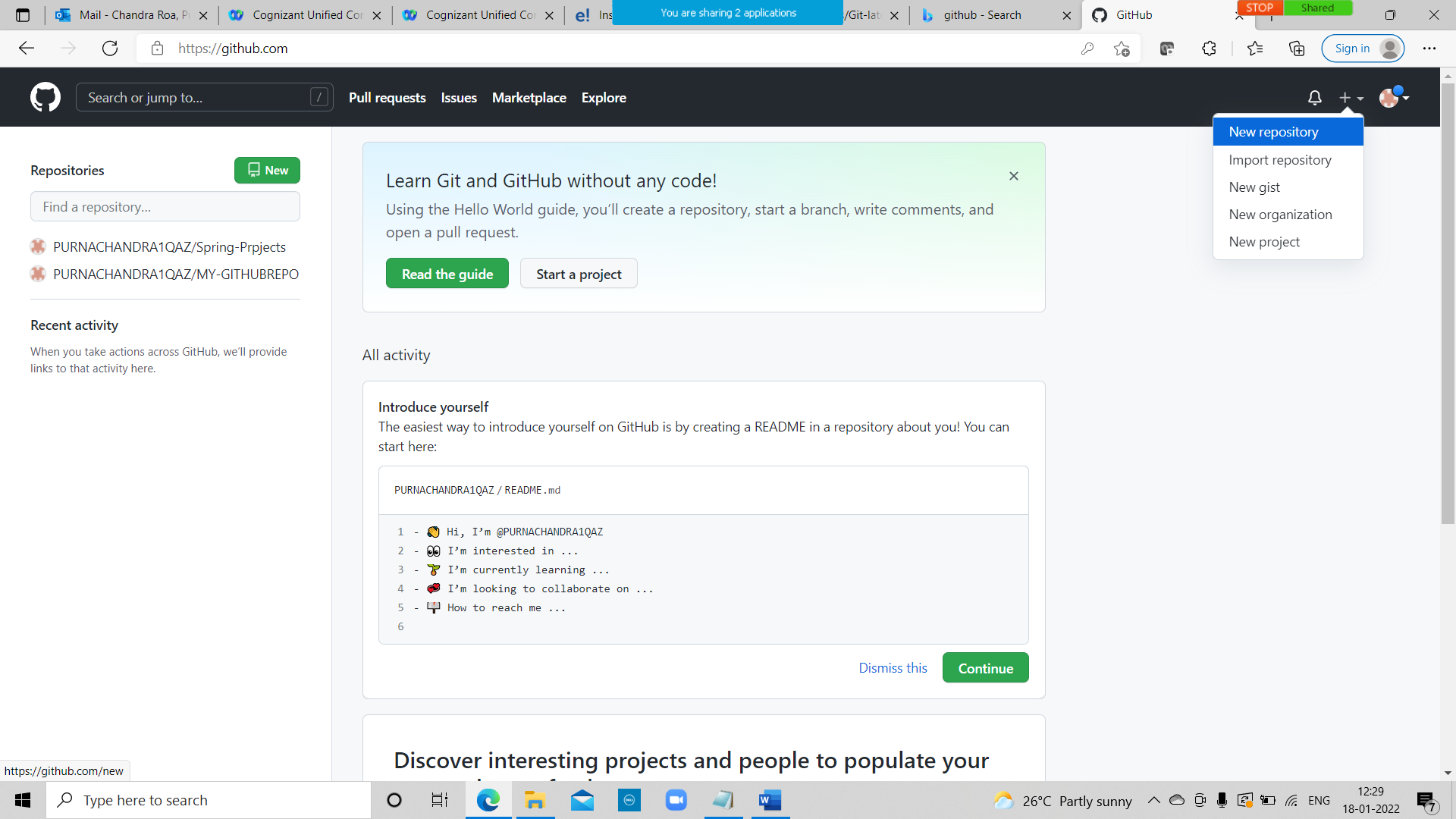
https://github.com/

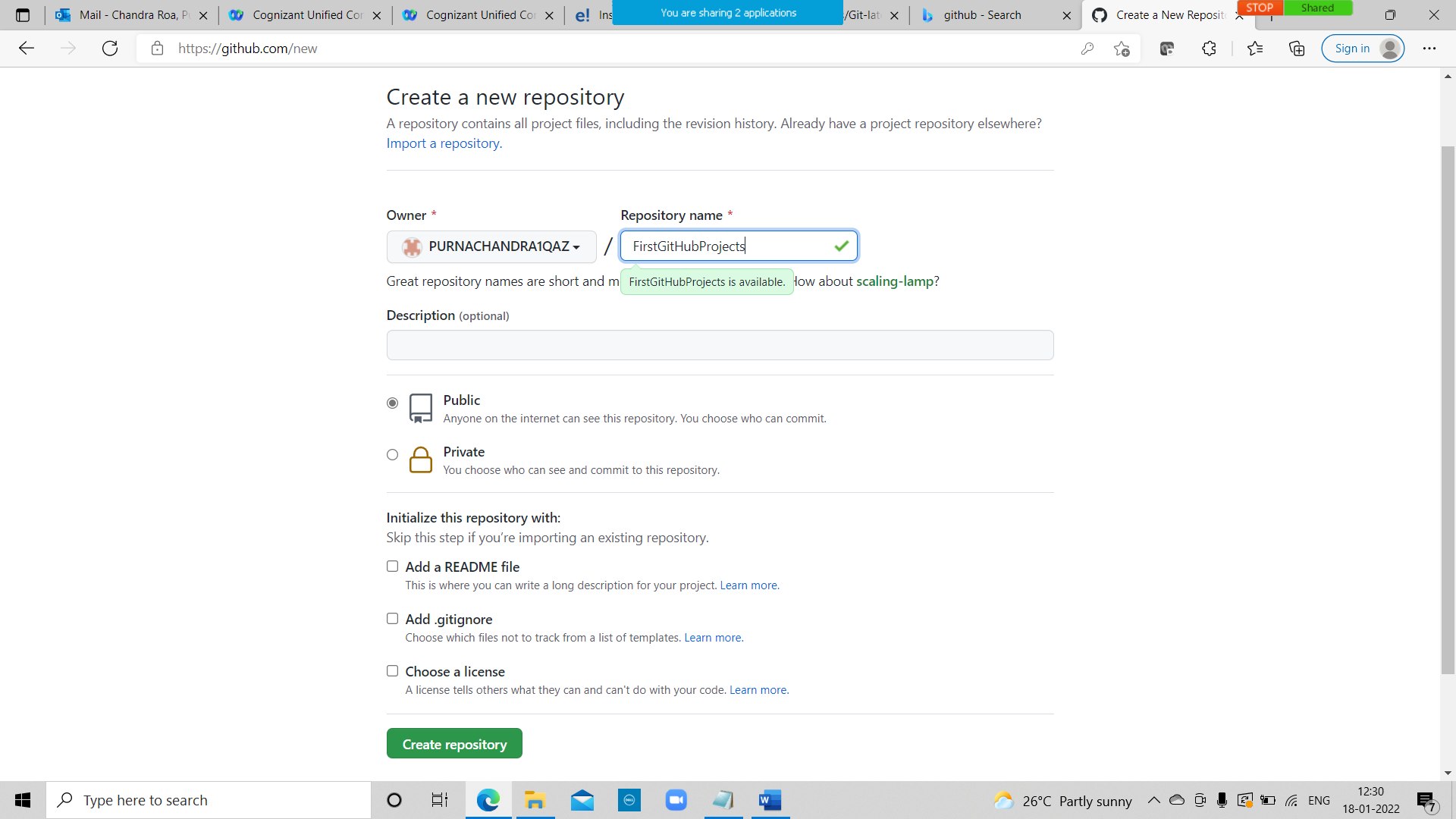




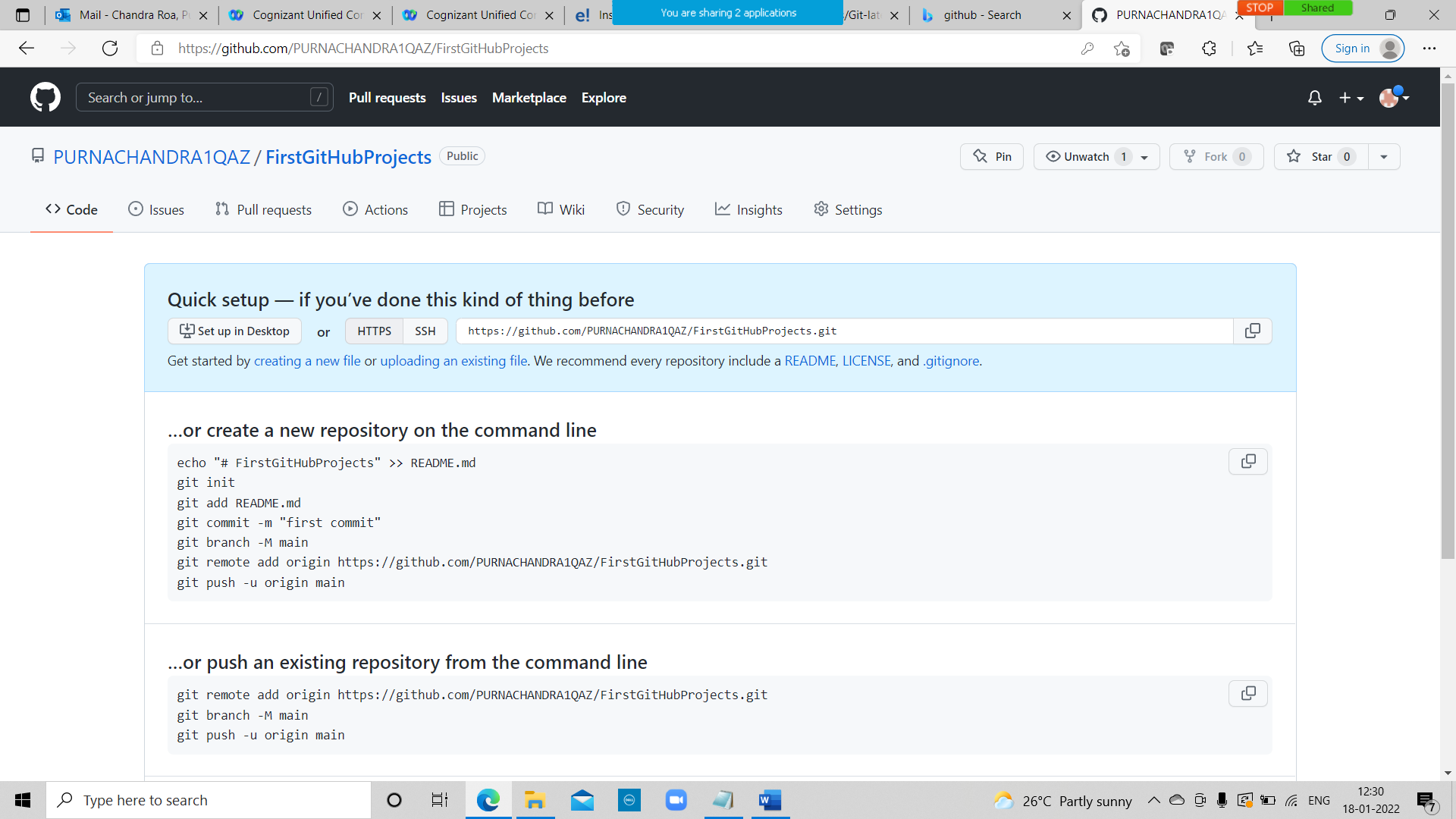




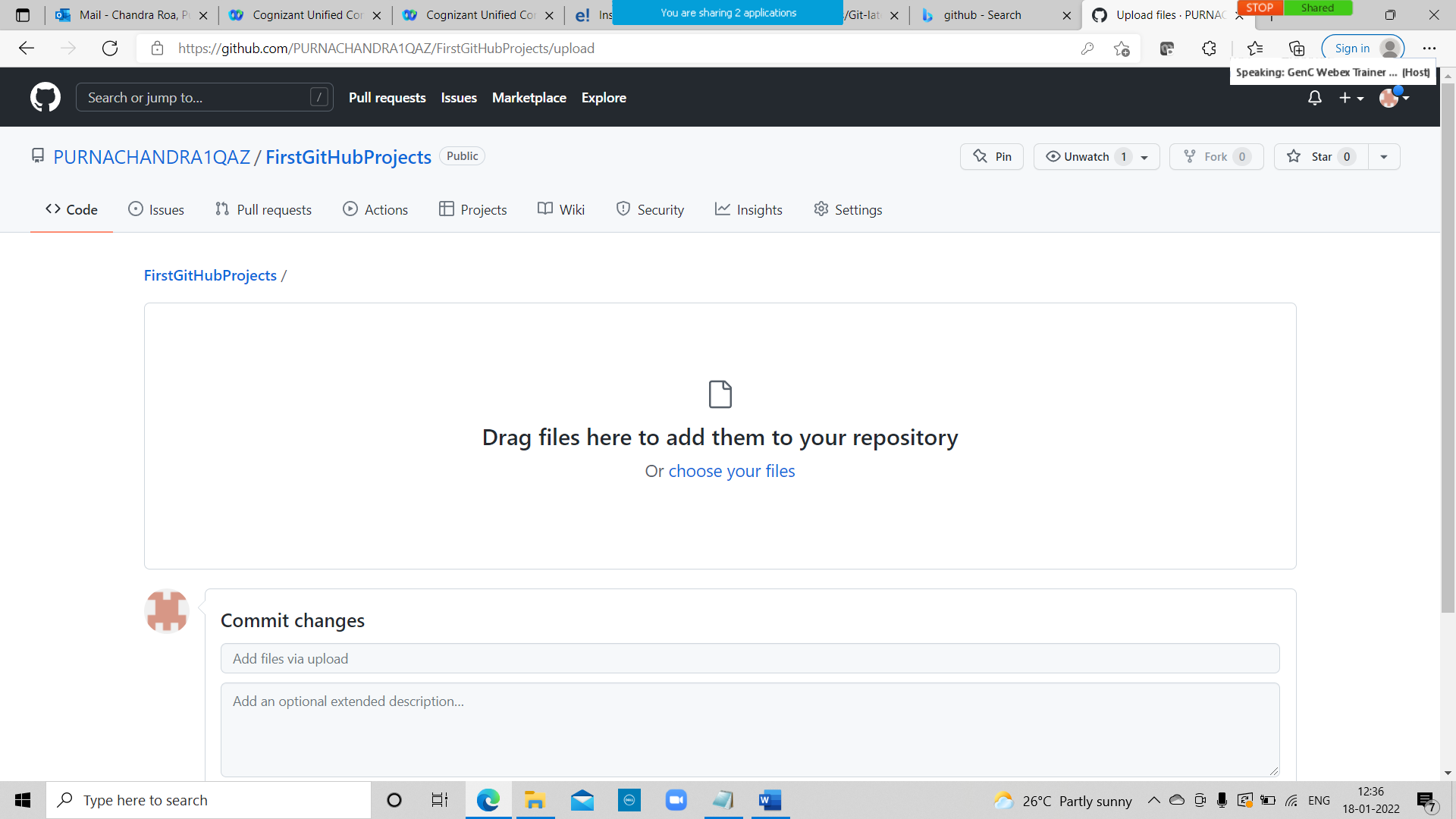


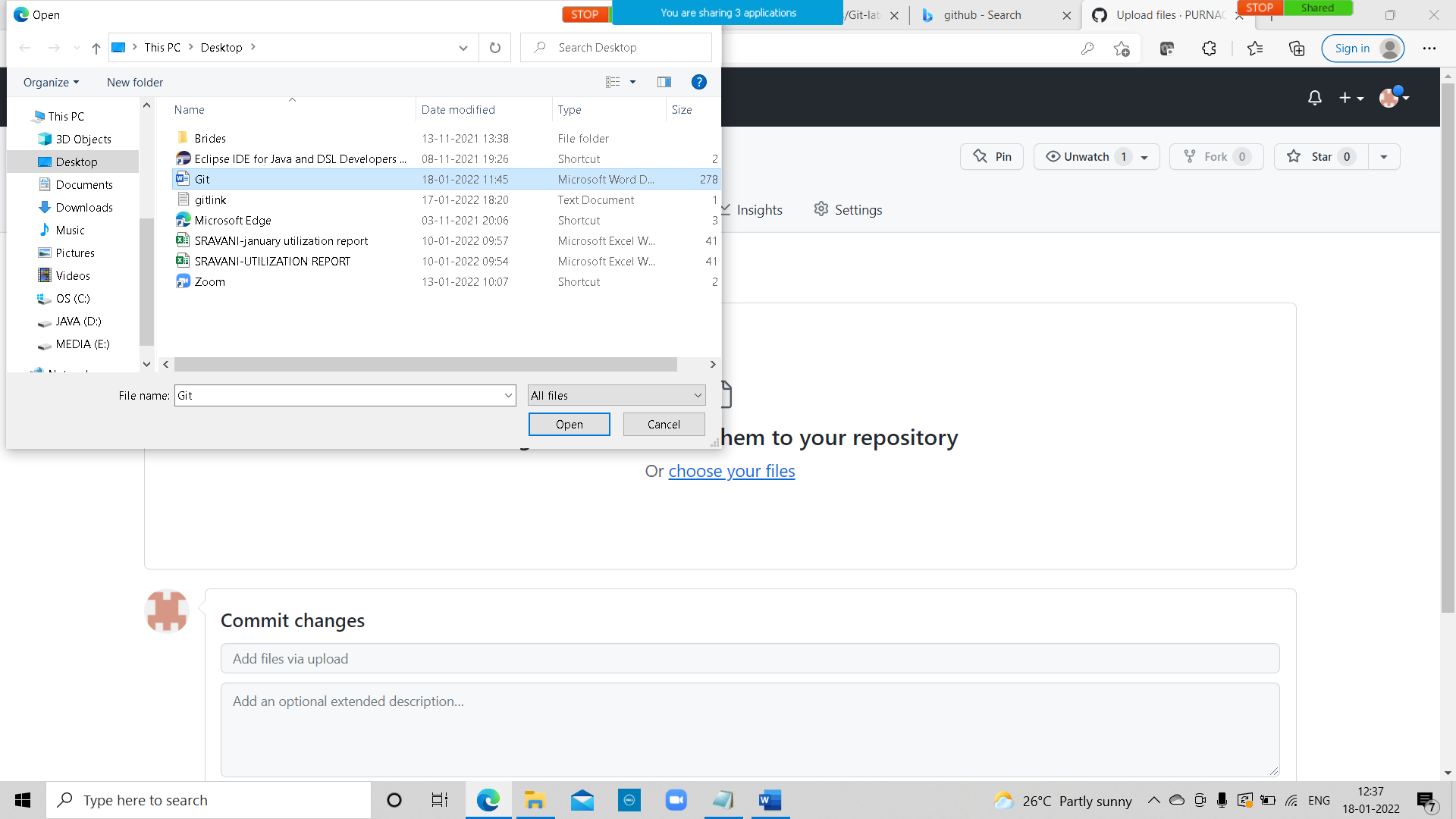


Click on create repository.

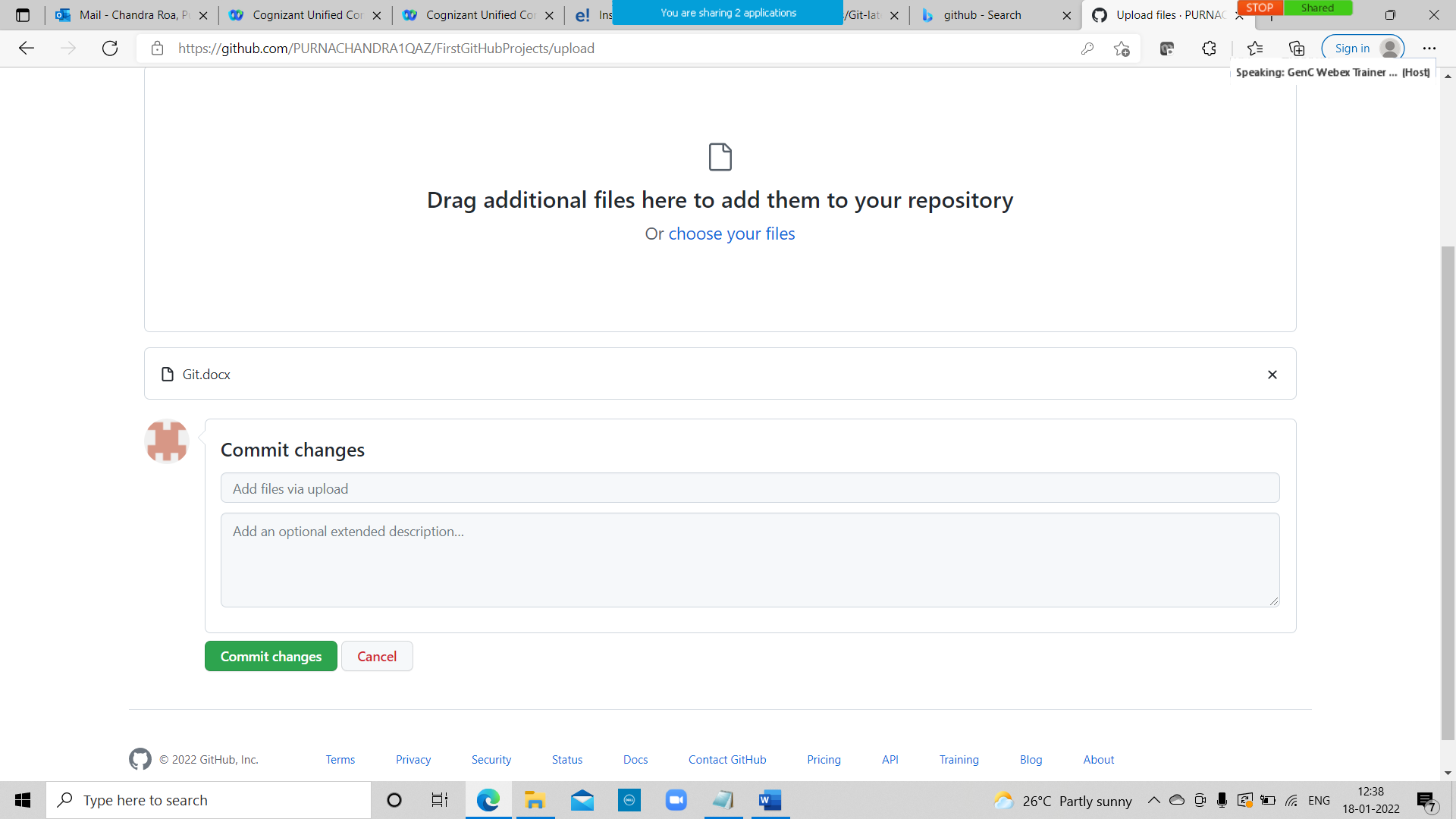


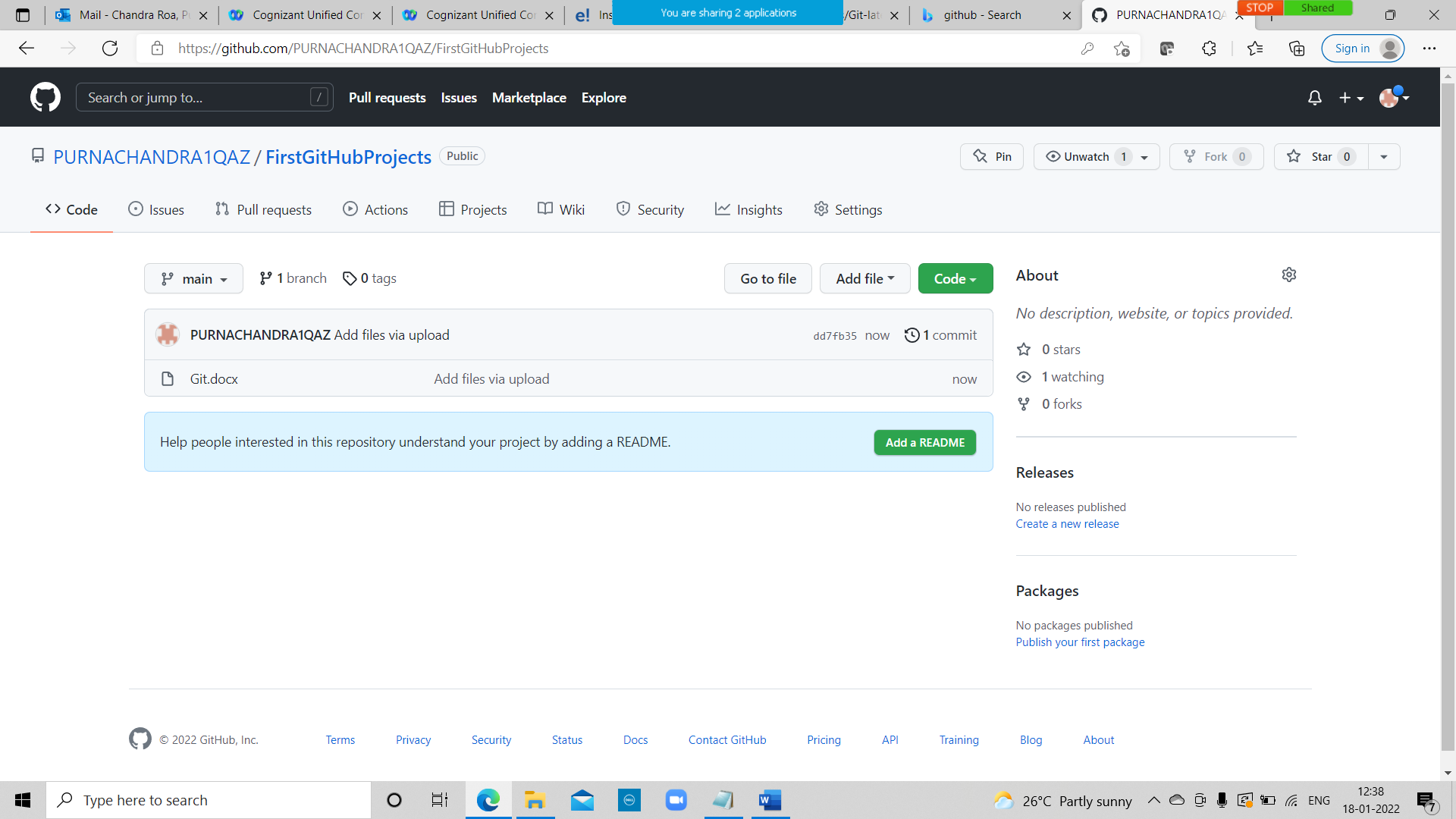
Click on upload an exisiting file





Click on commit changes

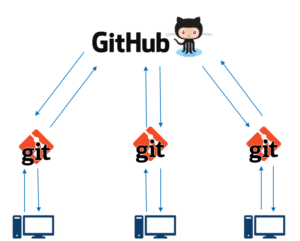




it is a file or code-sharing service to collaborate with different people.

GitHub is a highly used software that is typically used for version control. It is helpful when more than just one person is working on a project. Say for example, a software developer team wants to build a website and everyone has to update their codes simultaneously while working on the project. In this case, Github helps them to build a centralized repository where everyone can upload, edit, and manage the code files.

### ****Why is Github so popular?****

1. If you look at the image on the left, GitHub is a central repository and Git is a tool which allows you to create a local repository. Now people usually get confused between git and GitHub but its actually very different. Git is a version control tool that will allow you to perform all kinds of operations to fetch data from the central server or push data to it whereas GitHub is a core hosting platform for version control collaboration. GitHub is a company that allows you to host a central repository in a remote server.

Now let me list down the ways in which GitHub makes git simple:

* + GitHub provides you a beautiful visual interface which helps you to track or manage your version controlled projects locally.
  + Once you register on GitHub, you can connect with social network and build a strong profile.

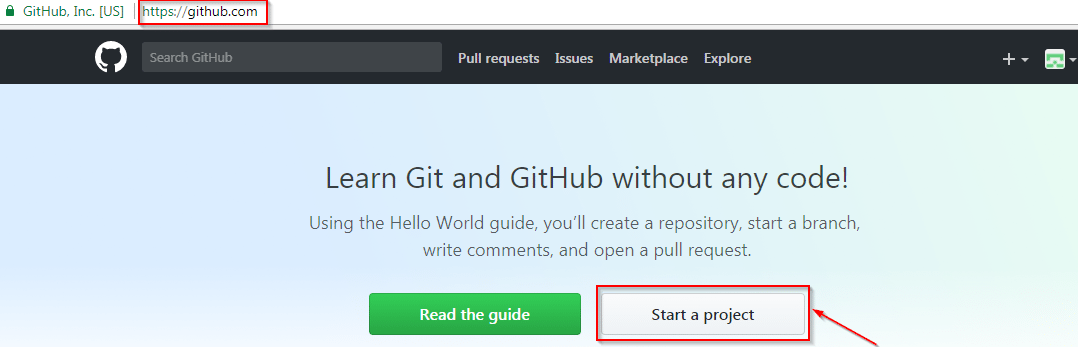
So let’s get started with GitHub.

## ****Step 2: How to create a GitHub Repository?****

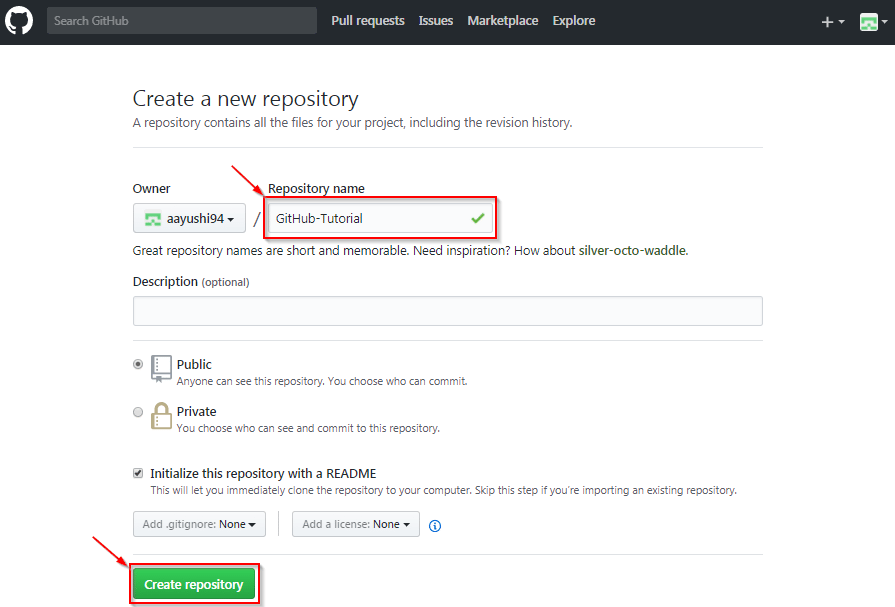
1. A repository is a storage space where your project lives.
2. It can be local to a folder on your computer, or it can be a storage space on GitHub  or another online host. You can keep code files, text files, images or any kind of a file in a repository.
3. You need a GitHub repository when you have done some changes and are ready to be uploaded. This GitHub repository acts as your remote repository.
4. So let me make your task easy, just follow these simple steps to create a GitHub repository:

* Go to the link: <https://github.com/> . Fill the sign up form and click on “Sign up for Github”.
* Click on “Start a new project”.

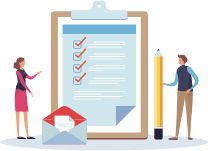
Refer to the below screenshot to get a better understanding.



* Enter any repository name and click on “Create Repository”. You can also give a description to your repository (optional).



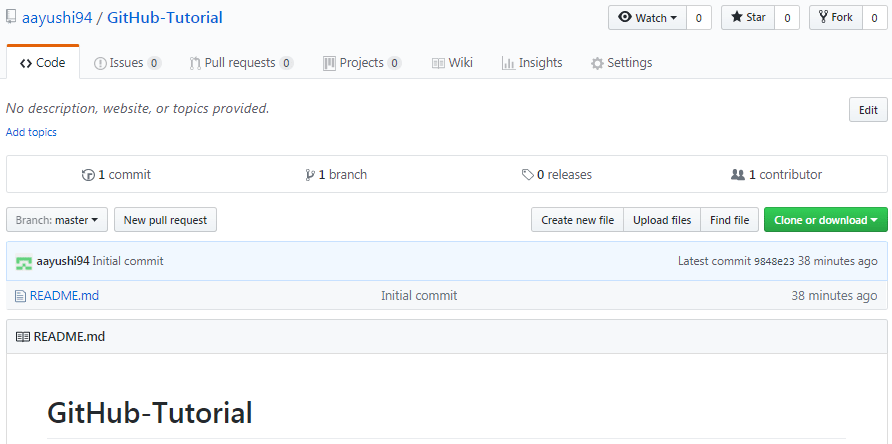
Now, if you noticed by default a GitHub repository is public which means that anyone can view the contents of this repository whereas in a private repository, you can choose who can view the content. Also, private repository is a paid version. Also, if you refer the above screenshot, initialize the repository with a README file. This file contains the description of the file and once you check this box, this will be the first file inside your repository.

[[](https://www.edureka.co/devops-engineer-training)](https://www.edureka.co/devops-engineer-training" \t "_blank)

### [DevOps Engineer Masters Program](https://www.edureka.co/devops-engineer-training" \t "_blank)

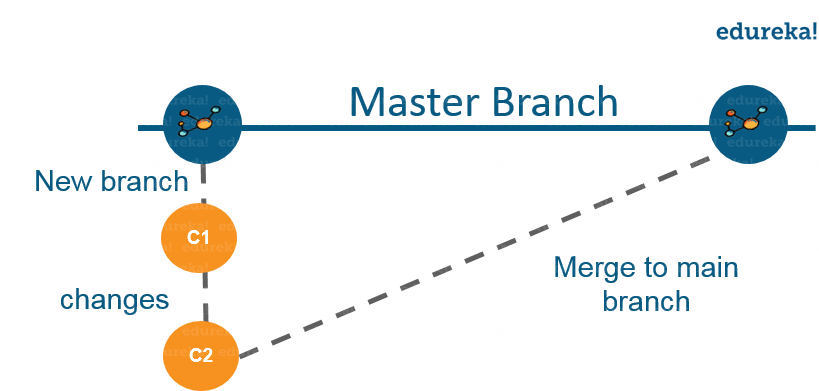
[Explore Curriculum](https://www.edureka.co/devops-engineer-training" \t "_blank)

Congratulations, your repository is successfully created! It will look like the below screenshot:



So now my central repository has been sucessfully created! Once this is done, you are ready to commit, pull, push and perform all the other operations. Now let’s move forward and understand branching in GitHub.

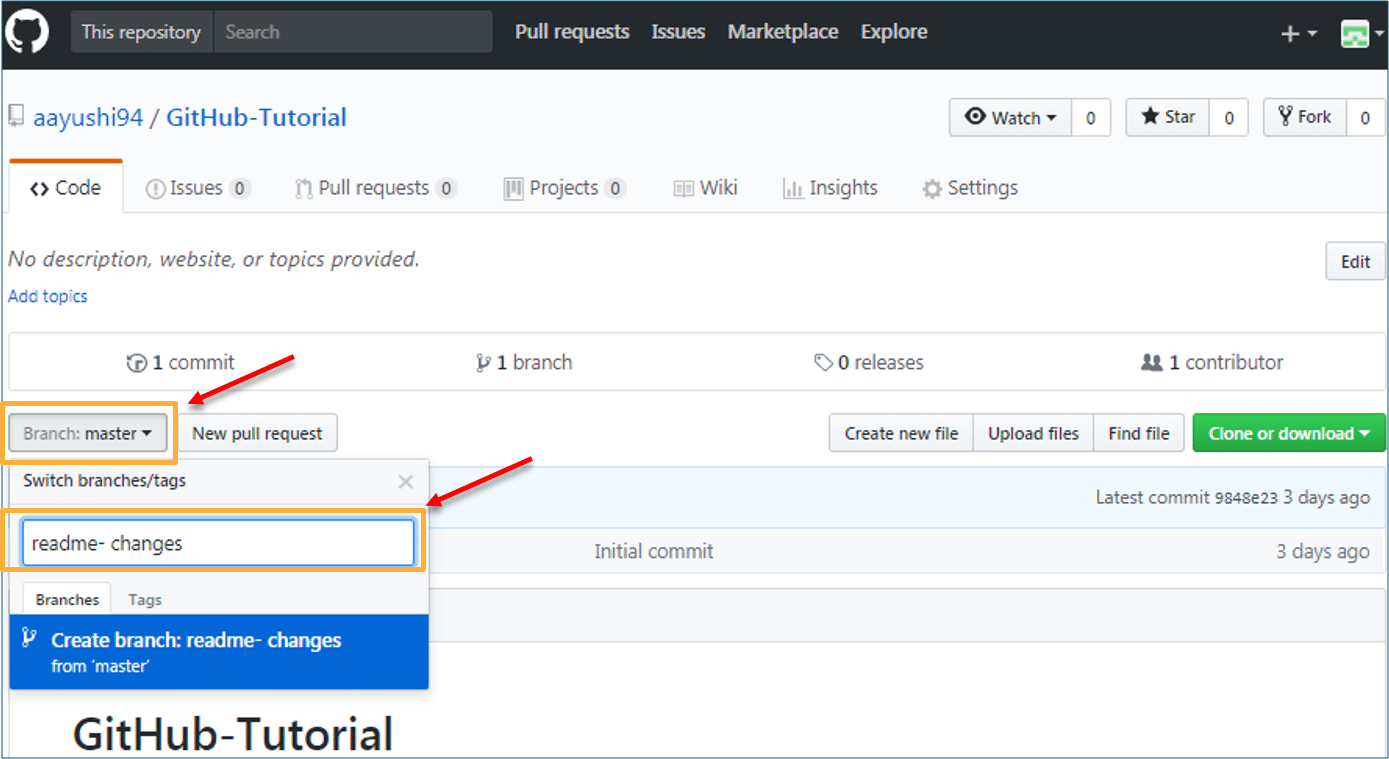
## ****Step 3: Create Branches and Perform Operations****



As depicted in the above image, there is a master/ production branch which has a new branch for testing. Under this branch, two set of changes are done and once it completed, it is merged back to the master branch. So this is how branching works!  
Let’s move ahead in ‘how to use GitHub’ blog, and learn how you can create a branch.

To create a branch in GitHub, follow the below steps:

* Click on the dropdown “Branch: master”
* As soon as you click on the branch, you can find an existing branch or you can create a new one. In my case, I am creating a new branch with a name “readme- changes”. Refer to the below screenshot for better understanding.



Once you have created a new branch, you have two branches in your repository now i.e. read-me (master branch) and readme- changes. The new branch is just the copy of master branch. So let’s perform some changes in our new branch and make it look different from the master branch.

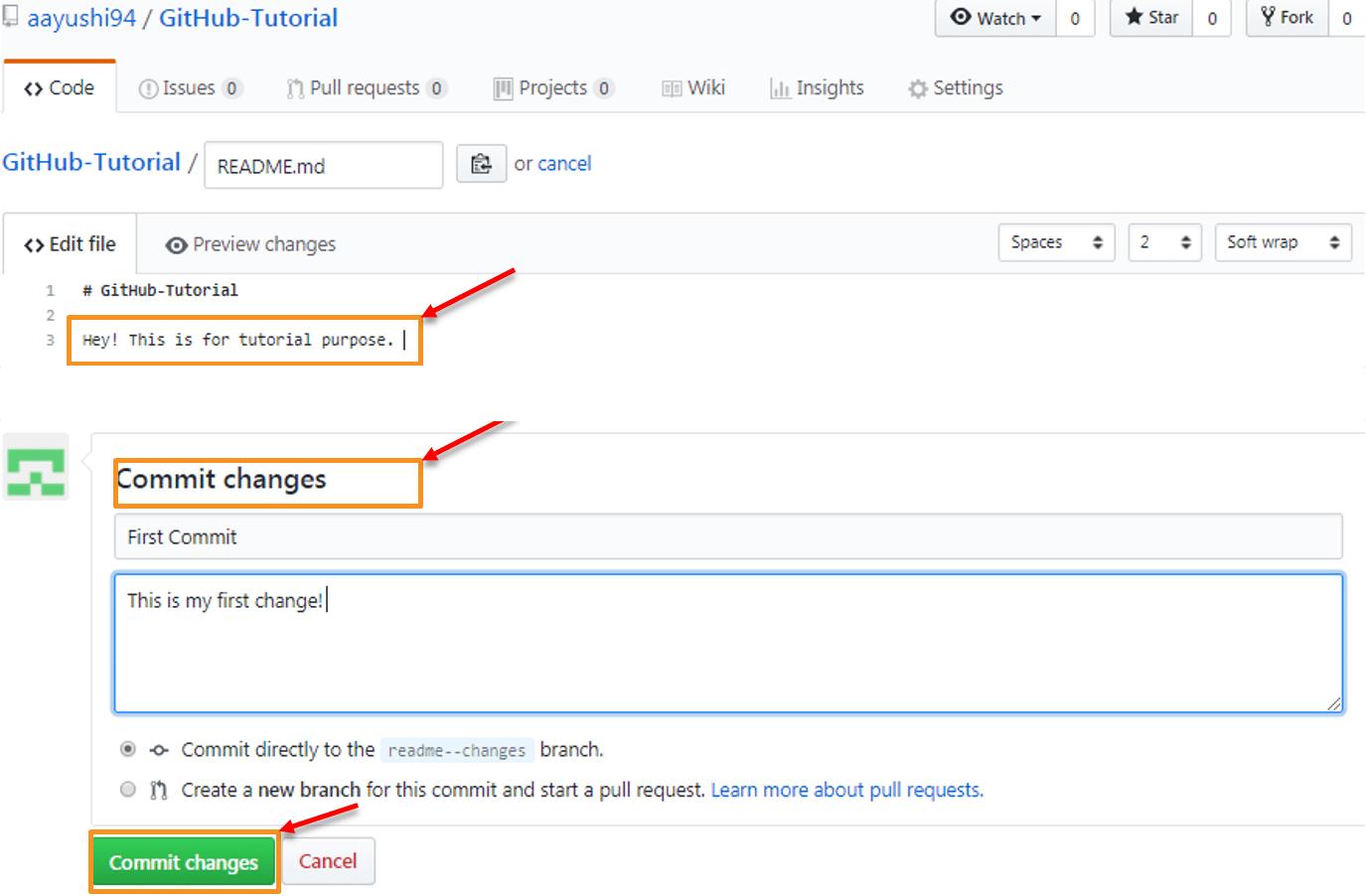
### ****How to use GitHub: Operations****

#### **Commit Command:**

This operation helps you to save the changes in your file. When you commit a file, you should always provide the message, just to keep in the mind the changes done by you. Though this message is not compulsory but it is always recommended so that it can differentiate the various versions or commits you have done so far to your repository. These commit messages maintain the history of changes which in turn help other contributors to understand the file better. Now let’s make our first commit, follow the below steps:

* Click on “readme- changes” file which we have just created.
* Click on the “edit” or a pencil icon in the righmost corner of the file.
* Once you click on that, an editor will open where you can type in the changes or anything.
* Write a commit message which identifies your changes.
* Click commit changes in the end.

Refer to the below screenshot for better understanding:



We have successfully made our first commit. Now this “readme- changes” file is different from the master branch. Next, let us see how can we open a pull request.

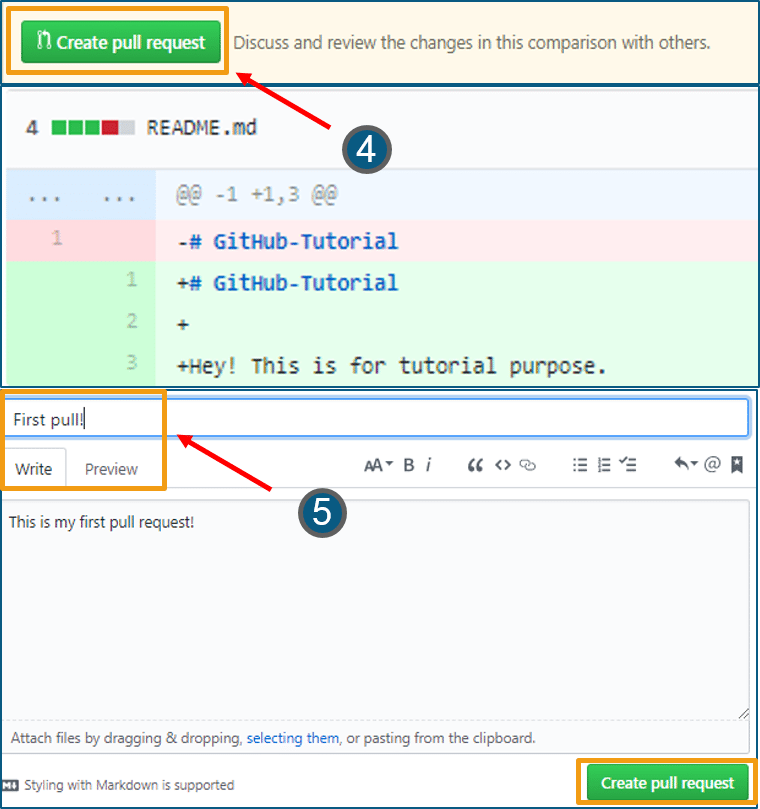
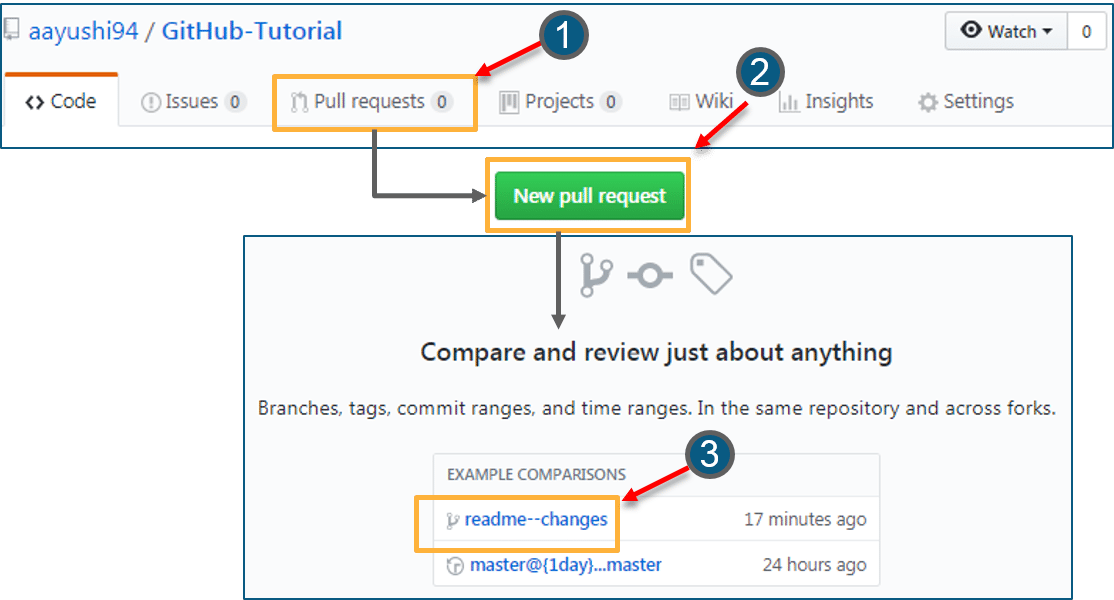
### [DevOps Engineer Masters Program](https://www.edureka.co/devops-engineer-training" \t "_blank)

## ****How to use GitHub | What is GitHub | Git and GitHub Tutorial | DevOps Training | Edureka****

#### **Pull Command**

Pull command is the most important command in GitHub. It tell the changes done in the file and request other contributors to view it as well as merge it with the master branch. Once the commit is done, anyone can pull the file and can start a discussion over it. Once its all done, you can merge the file. Pull command compares the changes which are done in the file and if there are any conflicts, you can manually resolve it. Now let us see different steps involved to pull request in GitHub.

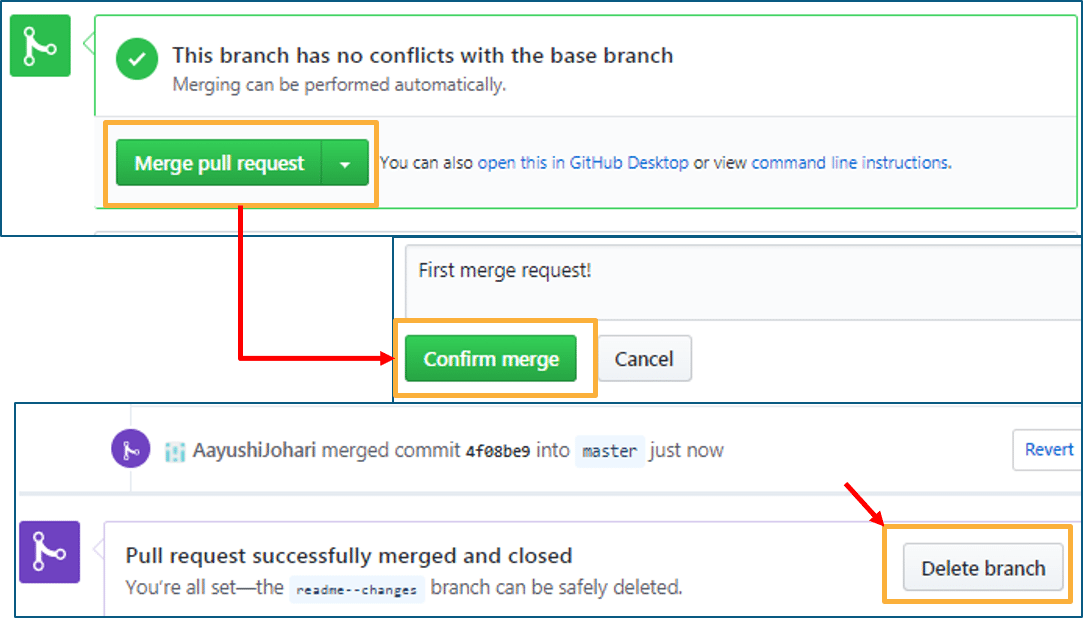
* Click the ‘Pull requests’ tab.
* Click ‘New pull request’.
* Once you click on pull request, select the branch and click ‘readme- changes’ file to view changes between the two files present in our repository.
* Click “Create pull request”.
* Enter any title, description to your changes and click on “Create pull request”. Refer to the below screenshots.

Next, let us move forward and see how can you merge your pull request.

#### **Merge Command**

Here comes the last command which merge the changes into the main master branch. We saw the changes in pink and green color, now let’s merge the “readme- changes” file with the master branch/ read-me. Go through the below steps to merge pull request.

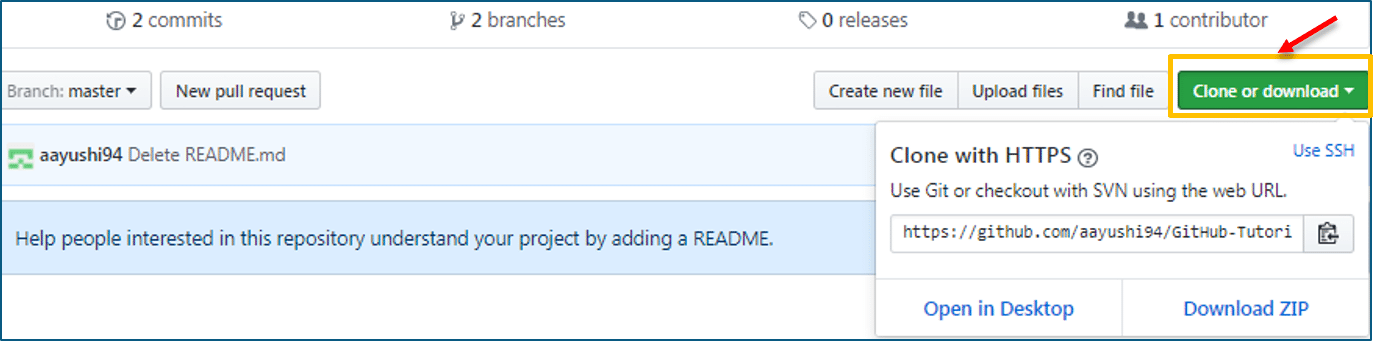
* Click on “Merge pull request” to merge the changes into master branch.
* Click “Confirm merge”.
* You can delete the branch once all the changes have been incorporated and if there are no conflicts. Refer to the below screenshots.



I hope you guys are trying these steps simultaneously while you are learning how to use GitHub. Next, let us move to our last topic in ‘how to use GitHub’ blog, i.e. Cloning and forking a GitHub repository.

## ****Step 4: Cloning and Forking GitHub Repository****

**Cloning:** Before I actually talk about cloning a GitHub repository, first let us understand why do we need to clone a repository. The answer is simple! Suppose you want to use some code which is present in a public repository, you can directly copy the contents by cloning or downloading. Refer to the below screenshot for a better understanding.



Cloning is really simple! In case you are facing any challenges on how to use GitHub, please comment your problems in the section below. Moving forward, let’s see what forking is.

**Forking:** First, let us talk about why do we need forking. Suppose, you need some code which is present in a public repository, under your repository and GitHub account. For this, we need to fork a repository.

Before we get started with forking, there are some important points which you should always keep in mind.

* Changes done to the original repository will be reflected back to the forked repository.
* If you make a change in forked repository, it will not b reflected to the original repository until and unless you have made a pull request.

Now let’s see how can you want to fork a repository. For that, follow the below steps:

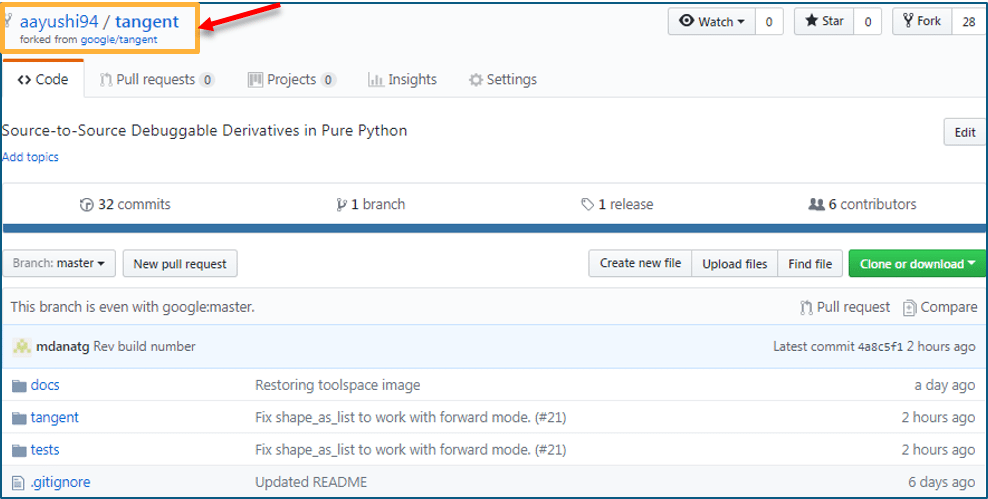
* Go to Explore and search for public repositories.
* Click “fork”. Note that this “tangent” repository is already forked 27 times and it is under “google”account. Refer the below image for better understanding.  
    
  

As soon as you click on “Fork”, it will take some time to fork the repository. Once done you will notice that the repository name is under your account. For reference, you can have a look at the below screenshot.

[[](https://www.edureka.co/devops-engineer-training)](https://www.edureka.co/devops-engineer-training" \t "_blank)

### [DevOps Engineer Masters Program](https://www.edureka.co/devops-engineer-training" \t "_blank)

[Weekday / Weekend BatchesSee Batch Details](https://www.edureka.co/devops-engineer-training" \t "_blank)



Congratulations! You have successfully forked an existing repository under your own account.

That’s all for this post, I hope you enjoyed it and got the solution for “how to use GitHub”.

## ****Git Tutorial****

Learning Git is as easy as using the tool. The motive of this Git tutorial blog is to omit this dilemma from your mind. I am sure with this Git tutorial blog, you will go brimming through all the concepts.

I hope that you have gone through the basic concepts and terminologies of Git and learned all about Version Control in my first blog of the Git Tutorial series. If you haven’t, please check out my [***previous blog***](https://www.edureka.co/blog/what-is-git/)to get a better understanding of Git.

In this Git Tutorial, you will learn:

* [Commands in Git](https://www.edureka.co/blog/git-tutorial/#commands_in_git)
* [Git operations](https://www.edureka.co/blog/git-tutorial/#operations_in_git)
* And some [tips and tricks](https://www.edureka.co/blog/git-tutorial/#tips_and_tricks) to manage your project effectively with Git.

Topics covered in this git tutorial will only gear you closer to become an ***[Edureka certified GitHub professional](https://www.edureka.co/git-github-sp" \t "_blank)***.

Before starting with the commands and operations let us first understand the primary motive of Git.

The motive of Git is to manage a project or a set of files as they change over time. Git stores this information in a data structure called a Git repository. The repository is the core of Git.

## ****How do I see my GIT repository?****

To be very clear, a Git repository is the directory where all of your project files and the related metadata resides.

Git records the current state of the project by creating a tree graph from the index. It is usually in the form of a Directed Acyclic Graph (DAG).

Before you go ahead, check out this video on Git tutorial to have better in-sight.

Now that you have understood what Git aims to achieve, let us go ahead with the operations and commands.

## ****How do I learn Git commands?****

A basic overview of how Git works:

* Create a “repository” (project) with a git hosting tool (like Bitbucket)
* Copy (or clone) the repository to your local machine
* Add a file to your local repo and “commit” (save) the changes
* “Push” your changes to your master branch
* Make a change to your file with a git hosting tool and commit
* “Pull” the changes to your local machine
* Create a “branch” (version), make a change, commit the change
* Open a “pull request”.
* “Merge” your branch to the master branch

## ****What is the difference between Git Shell and Git Bash?****

Git Bash and Git Shell are two different command line programs which allow you to interact with the underlying Git program. Bash is a Linux-based command line while Shell is a native Windows command line.

## ****Git Tutorial – A few Operations & Commands****

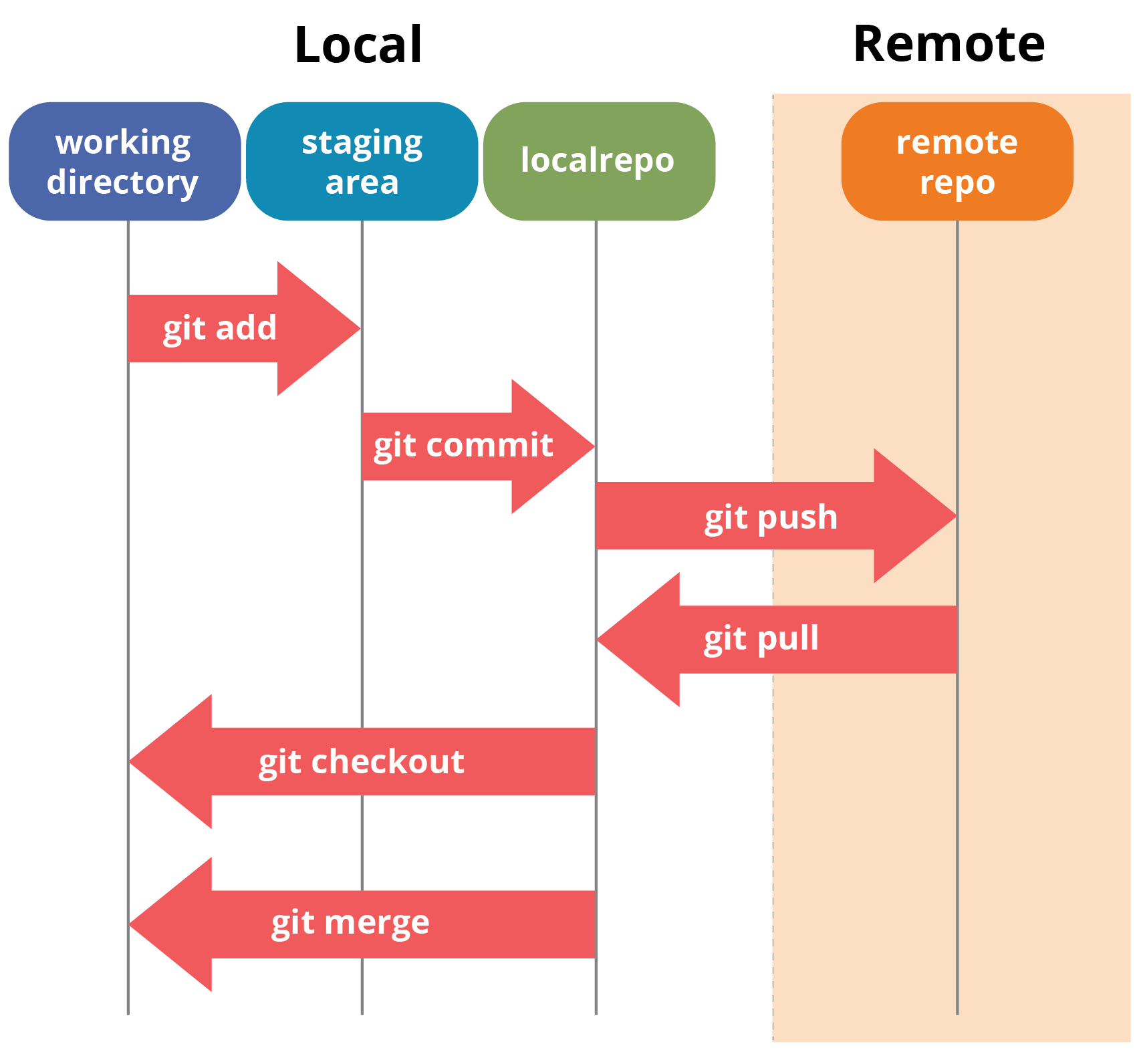
Some of the basic operations in Git are:

1. Initialize
2. Add
3. Commit
4. Pull
5. Push

Some advanced Git operations are:

1. Branching
2. Merging
3. Rebasing

Let me first give you a brief idea about how these operations work with the Git repositories. Take a look at the architecture of Git below:



If you understand the above diagram well and good, but if you don’t, you need not worry, I will be explaining these operations in this Git Tutorial one by one. Let us begin with the basic operations.

You need to install Git on your system first. If you need help with the installation, [***click here***](https://www.edureka.co/blog/install-git/).

### ****What is Git Bash used for?****

This Git Bash Tutorial focuses on the commands and operations that can be used on Git Bash.

Note: The commands cannot be executed on GitHub. You can check out [*this blog on GitHub tutorial*](https://www.edureka.co/blog/how-to-use-github/) for reference.

### ****How do I navigate Git Bash?****

After installing Git in your Windows system, just open your folder/directory where you want to store all your project files; right click and select ‘***Git Bash here***’.

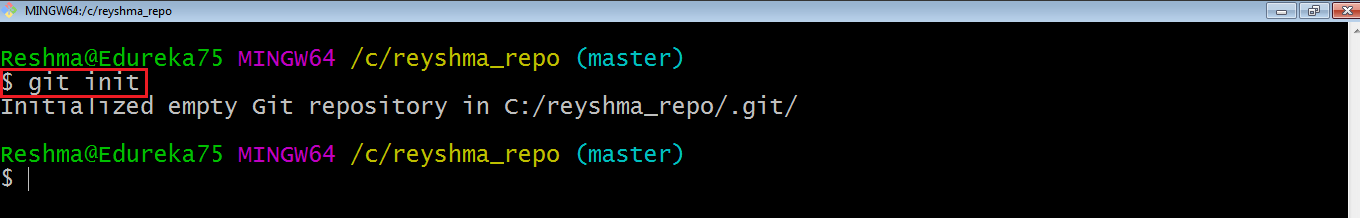


This will open up Git Bash terminal where you can enter commands to perform various Git operations.

Now, the next task is to initialize your repository.

## ****Initialize****

In order to do that, we use the command **git init.**Please refer to the below screenshot.



**git init** creates an empty Git repository or re-initializes an existing one. It basically creates a**.git** directory with sub directories and template files. Running a **git init** in an existing repository will not overwrite things that are already there. It rather picks up the newly added templates.

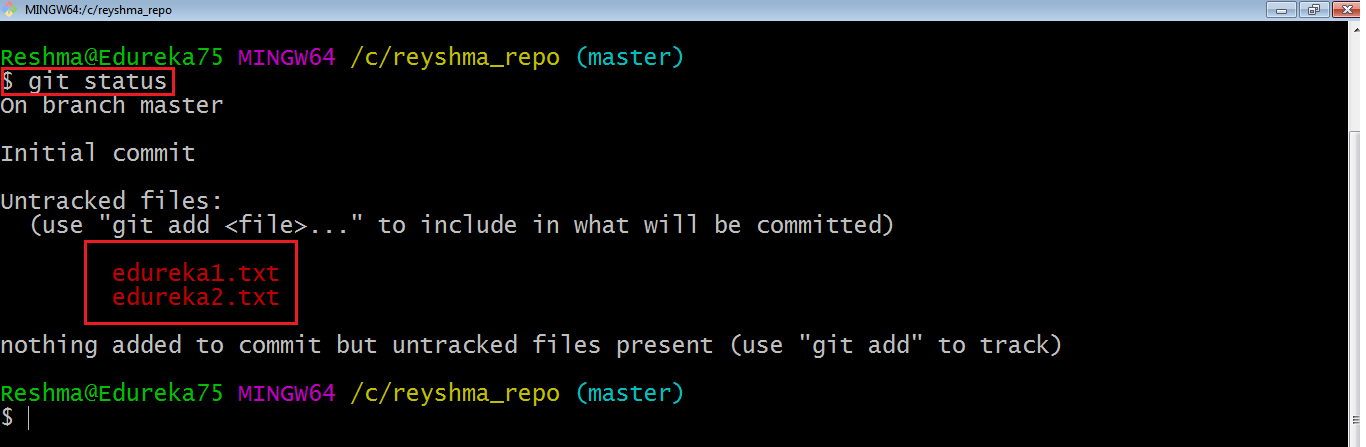
Now that my repository is initialized, let me create some files in the directory/repository. For e.g. I have created two text files namely edureka1.txt and edureka2.txt.

Let’s see if these files are in my index or not using the command **git status**. The index holds a snapshot of the content of the working tree/directory, and this snapshot is taken as the contents for the next change to be made in the local repository.

**Git status**

The **git status**command lists all the modified files which are ready to be added to the local repository.

Let us type in the command to see what happens:



This shows that I have two files which are not added to the index yet. This means I cannot commit changes with these files unless I have added them explicitly in the index.

**Add**

This command updates the index using the current content found in the working tree and then prepares the content in the staging area for the next commit.

Thus, after making changes to the working tree, and before running the **commit** command, you must use the **add** command to add any new or modified files to the index. For that, use the commands below:

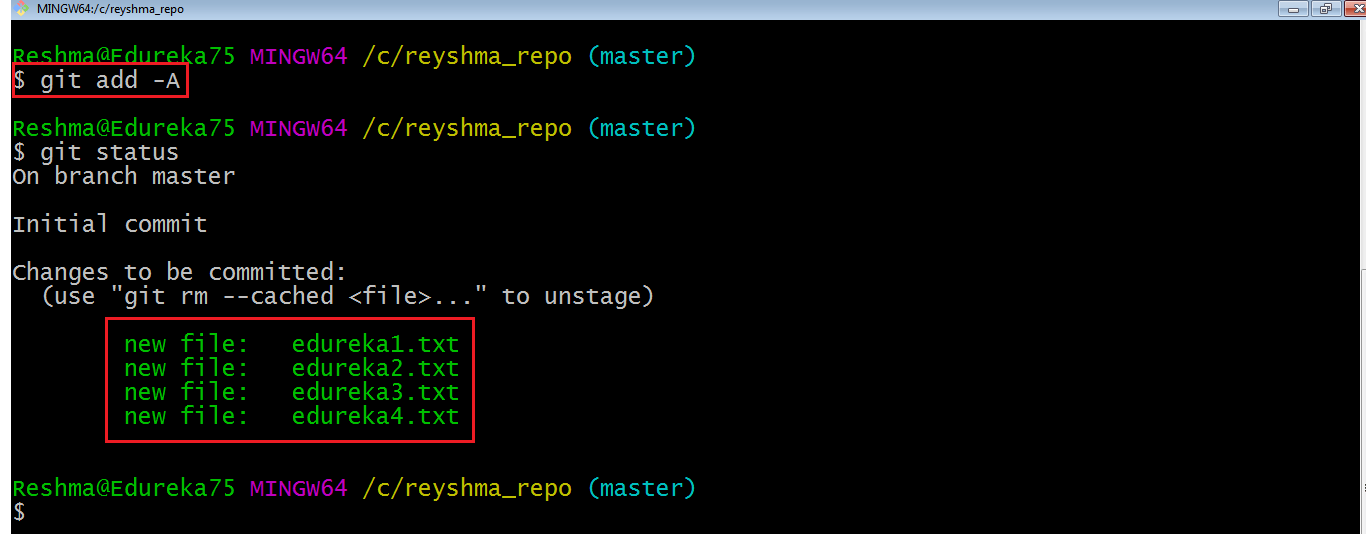
**git add <directory>**

or

**git add <file>**

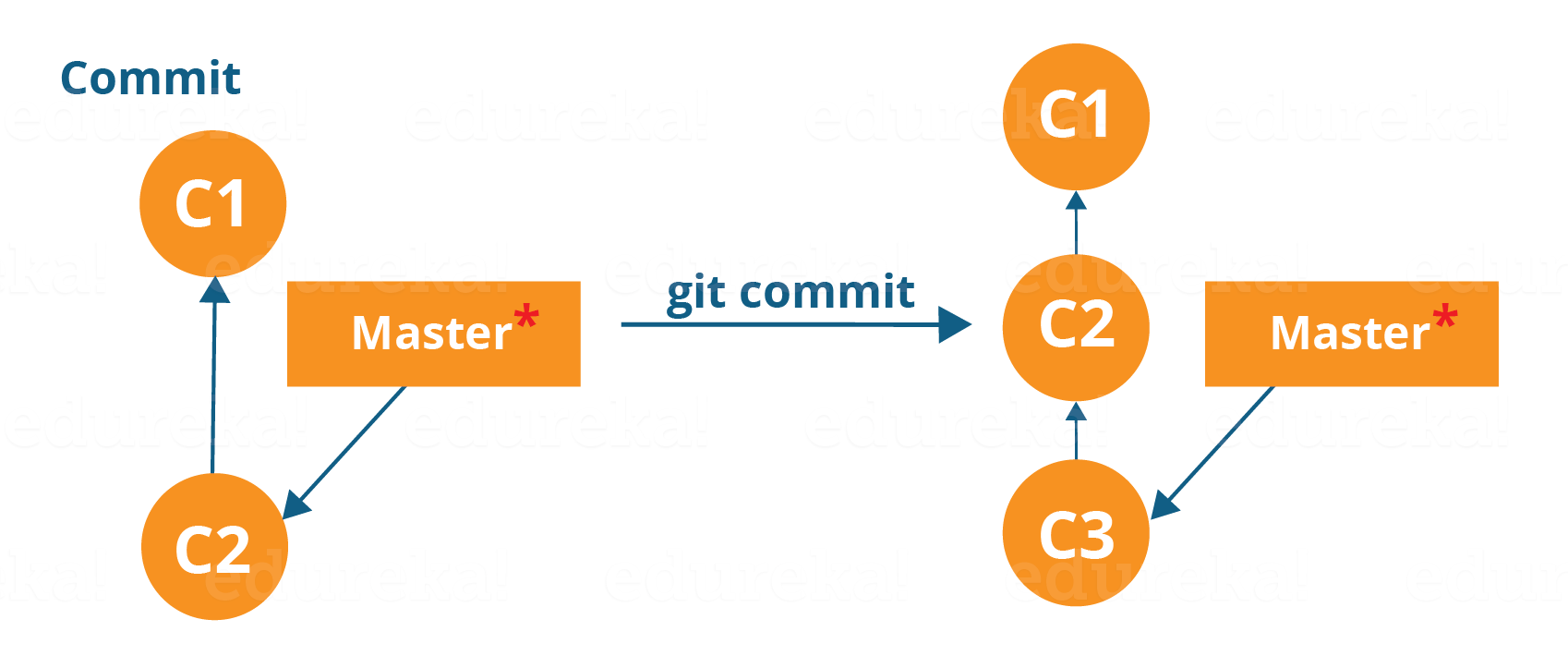
Let me demonstrate the **git add** for you so that you can understand it better.

I have created two more files edureka3.txt and edureka4.txt. Let us add the files using the command **git add -A**. This command will add all the files to the index which are in the directory but not updated in the index yet.



Now that the new files are added to the index, you are ready to commit them.

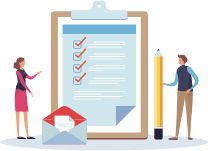
**Commit**

It refers to recording snapshots of the repository at a given time. Committed snapshots will never change unless done explicitly. Let me explain how commit works with the diagram below:

Here, C1 is the initial commit, i.e. the snapshot of the first change from which another snapshot is created with changes named C2. Note that the master points to the latest commit.

Now, when I commit again, another snapshot C3 is created and now the master points to C3 instead of C2.

Git aims to keep commits as lightweight as possible. So, it doesn’t blindly copy the entire directory every time you commit; it includes commit as a set of changes, or “delta” from one version of the repository to the other. In easy words, it only copies the changes made in the repository.

[[](https://www.edureka.co/devops-certification-training)](https://www.edureka.co/devops-certification-training" \t "_blank)

### [DevOps Certification Training Course](https://www.edureka.co/devops-certification-training" \t "_blank)

[Explore Curriculum](https://www.edureka.co/devops-certification-training" \t "_blank)

You can commit by using the command below:

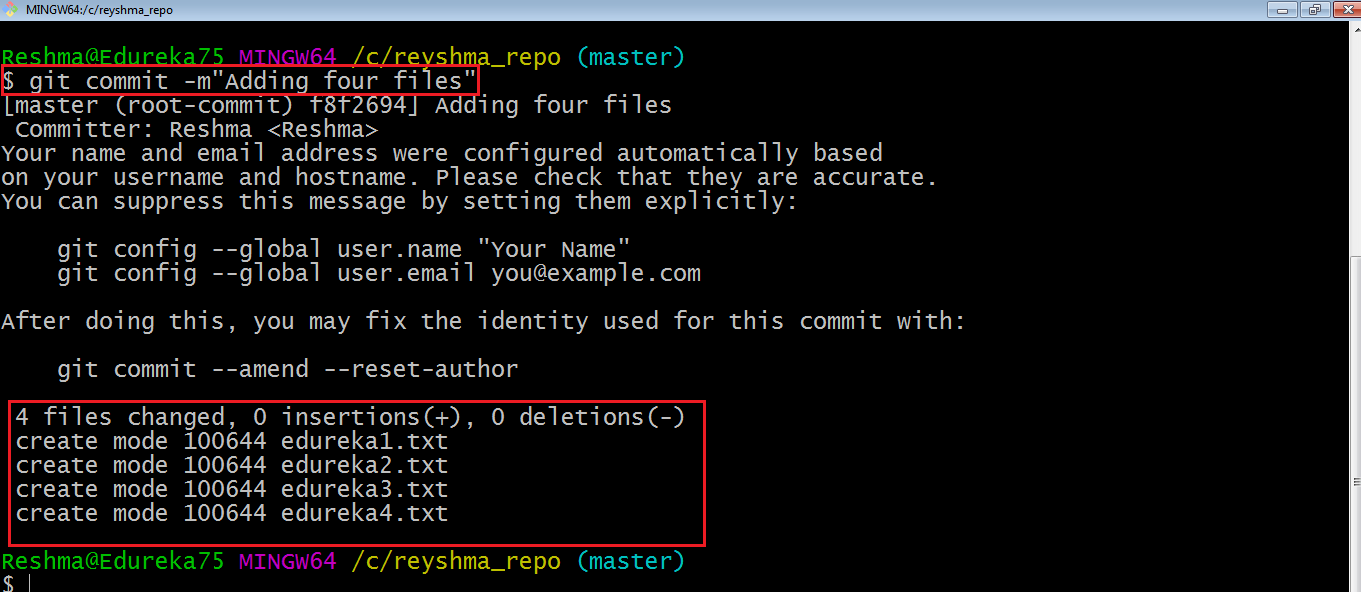
**git commit**

This will commit the staged snapshot and will launch a text editor prompting you for a commit message.

Or you can use:

**git commit -m “<message>”**

Let’s try it out.



As you can see above, the **git commit** command has committed the changes in the four files in the local repository.

Now, if you want to commit a snapshot of all the changes in the working directory at once, you can use the command below:

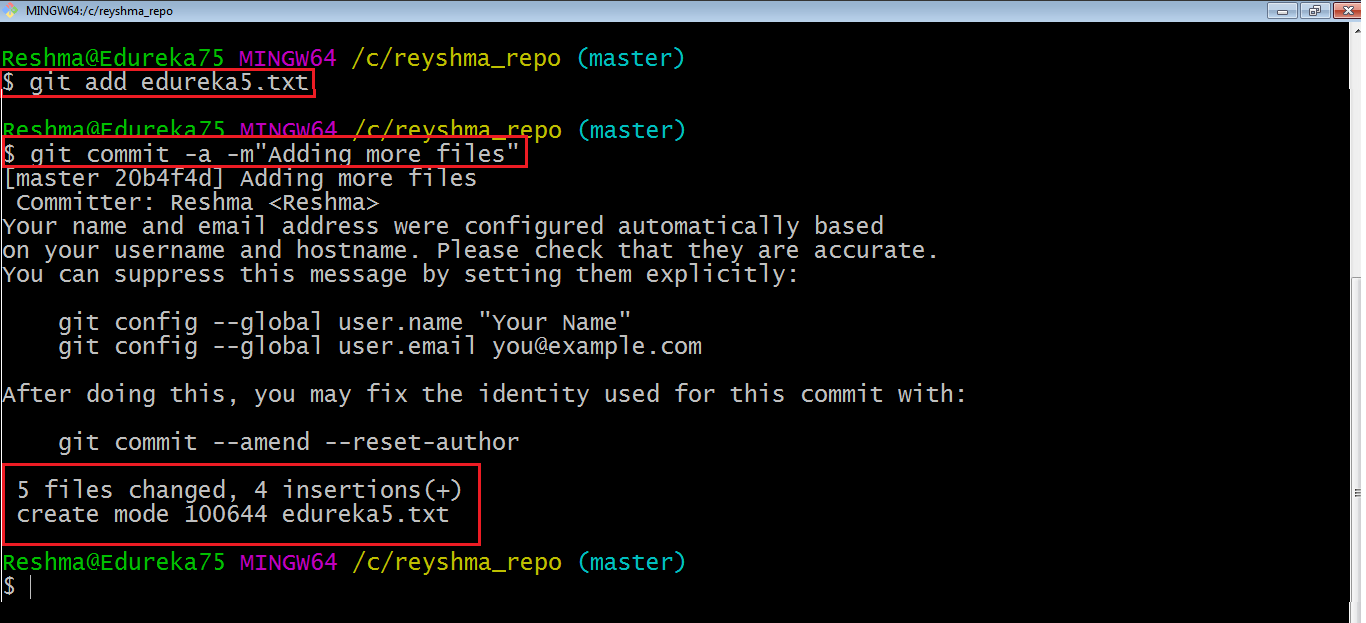
**git commit -a**

I have created two more text files in my working directory viz. edureka5.txt and edureka6.txt but they are not added to the index yet.

I am adding edureka5.txt using the command:

**git add edureka5.txt**

I have added edureka5.txt to the index explicitly but not edureka6.txt and made changes in the previous files. I want to commit all changes in the directory at once. Refer to the below snapshot.



This command will commit a snapshot of all changes in the working directory but only includes modifications to tracked files i.e. the files that have been added with **git add** at some point in their history. Hence, edureka6.txt was not committed because it was not added to the index yet. But changes in all previous files present in the repository were committed, i.e. edureka1.txt, edureka2.txt, edureka3.txt, edureka4.txt and edureka5.txt.  
Now I have made my desired commits in my local repository.

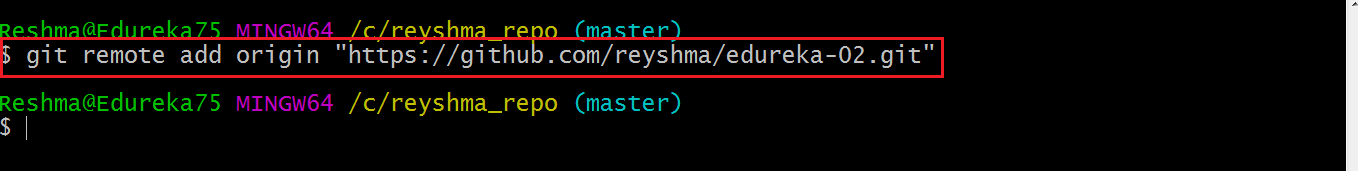
Note that before you affect changes to the central repository you should always pull changes from the central repository to your local repository to get updated with the work of all the collaborators that have been contributing in the central repository. For that we will use the **pull** command.

## ****Pull****

The **git pull** command fetches changes from a remote repository to a local repository. It merges upstream changes in your local repository, which is a common task in Git based collaborations.

But first, you need to set your central repository as origin using the command:

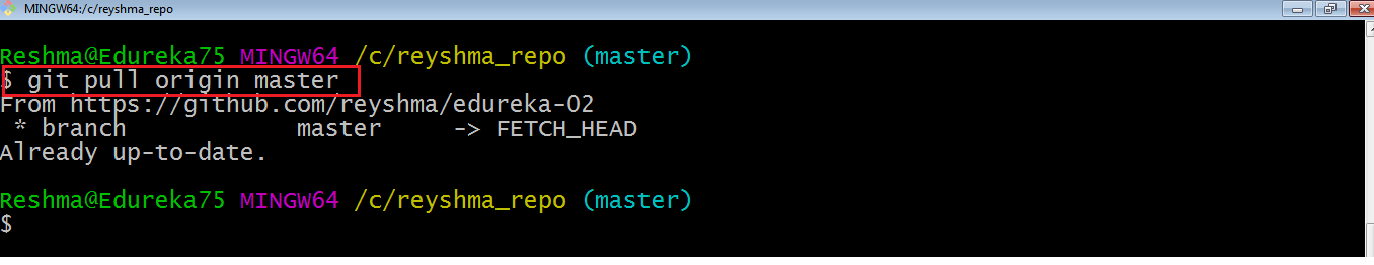
**git remote add origin <link of your central repository>**



Now that my origin is set, let us extract files from the origin using pull. For that use the command:

**git pull origin master**

This command will copy all the files from the master branch of remote repository to your local repository.



Since my local repository was already updated with files from master branch, hence the message is Already up-to-date. Refer to the screen shot above.

***Note:*** One can also try pulling files from a different branch using the following command:

***git pull origin <branch-name>***

Your local Git repository is now updated with all the recent changes. It is time you make changes in the central repository by using the**push** command.

**Push**

This command transfers commits from your local repository to your remote repository. It is the opposite of pull operation.

Pulling imports commits to local repositories whereas pushing exports commits to the remote repositories .

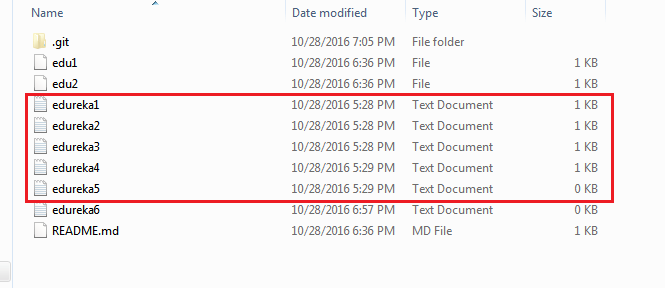
The use of **git push** is to publish your local changes to a central repository. After you’ve accumulated several local commits and are ready to share them with the rest of the team, you can then push them to the central repository by using the following command:

**git push <remote>**

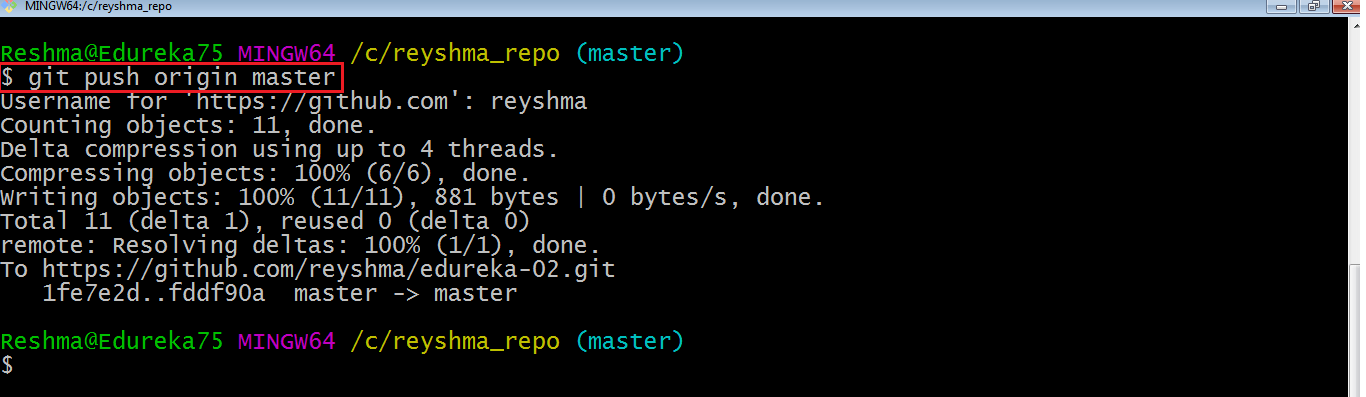
**Note** : This remote refers to the remote repository which had been set before using the pull command.

This pushes the changes from the local repository to the remote repository along with all the necessary commits and internal objects. This creates a local branch in the destination repository.

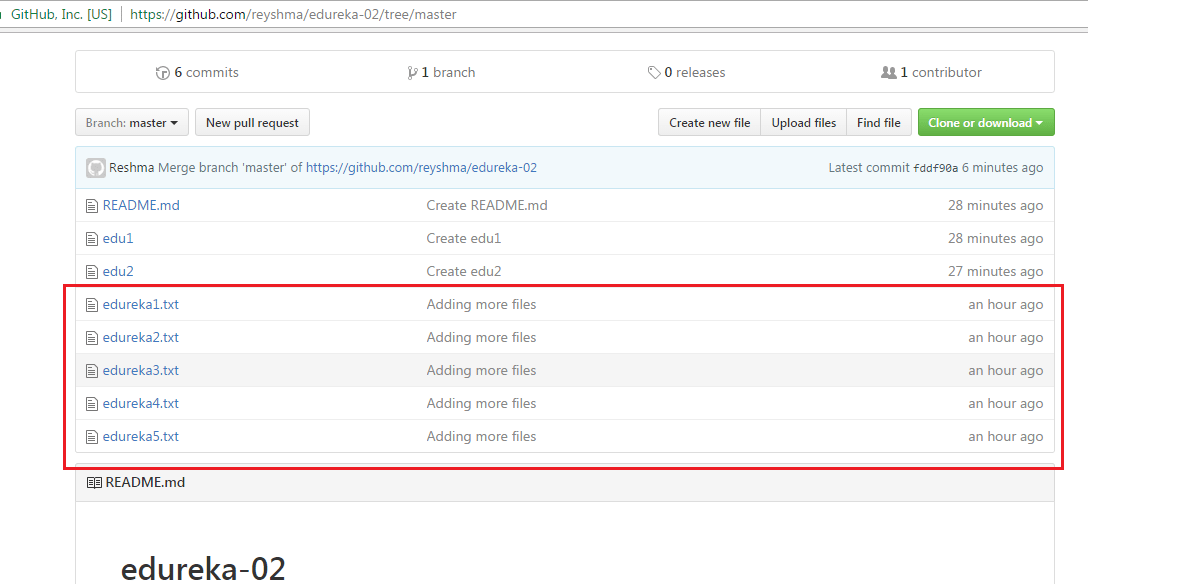
Let me demonstrate it for you.



The above files are the files which we have already committed previously in the commit section and they are all “push-ready“. I will use the command **git push origin master** to reflect these files in the master branch of my central repository.



Let us now check if the changes took place in my central repository.



Yes, it did. :-)

To prevent overwriting, Git does not allow push when it results in a non-fast forward merge in the destination repository.

**Note**: A non-fast forward merge means an upstream merge i.e. merging with ancestor or parent branches from a child branch.

To enable such merge, use the command below:

**git push <remote> –force**

The above command forces the push operation even if it results in a non-fast forward merge.

At this point of this Git Tutorial, I hope you have understood the basic commands of Git. Now, let’s take a step further to learn branching and merging in Git.

## ****Branching****

Branches in Git are nothing but pointers to a specific commit. Git generally prefers to keep its branches as lightweight as possible.

There are basically two types of branches viz. **local branches** and **remote tracking branches**.

### [DevOps Certification Training Course](https://www.edureka.co/devops-certification-training" \t "_blank)

A local branch is just another path of your working tree. On the other hand, remote tracking branches have special purposes. Some of them are:

* They link your work from the local repository to the work on central repository.
* They automatically detect which remote branches to get changes from, when you use **git pull**.

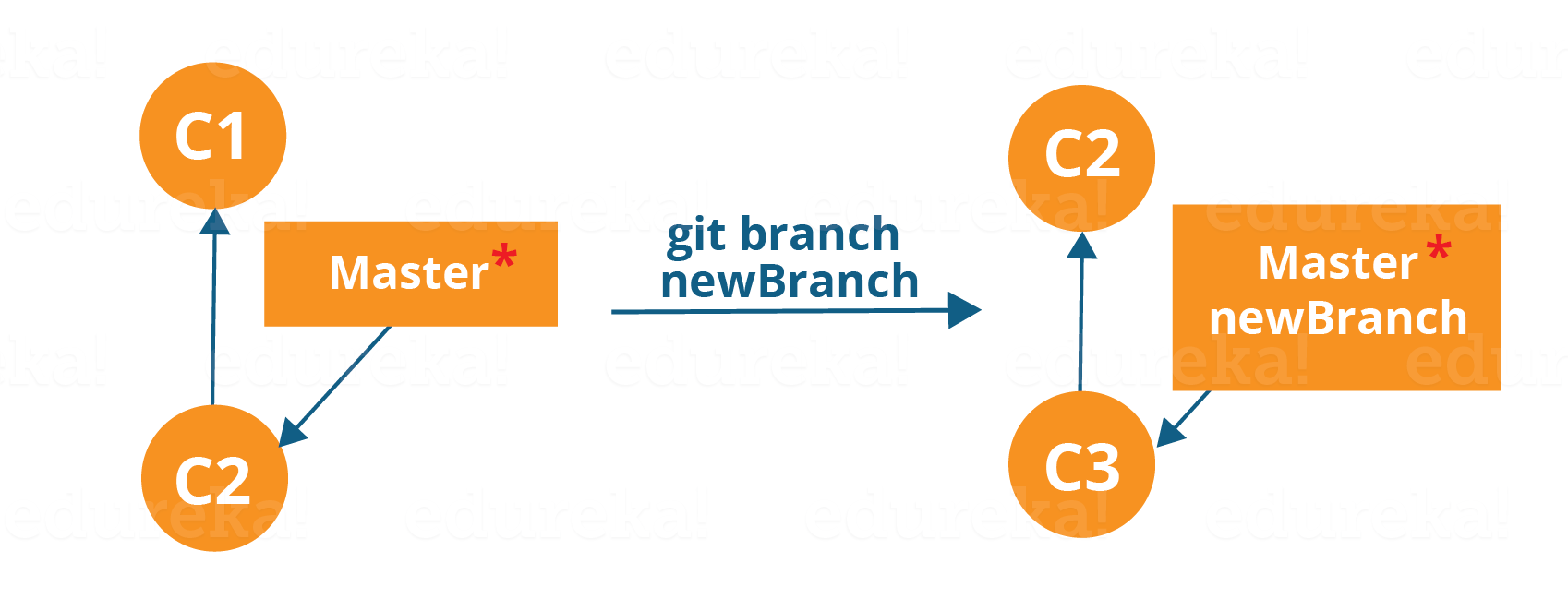
You can check what your current branch is by using the command:

**git branch**

The one mantra that you should always be chanting while branching is “branch early, and branch often”

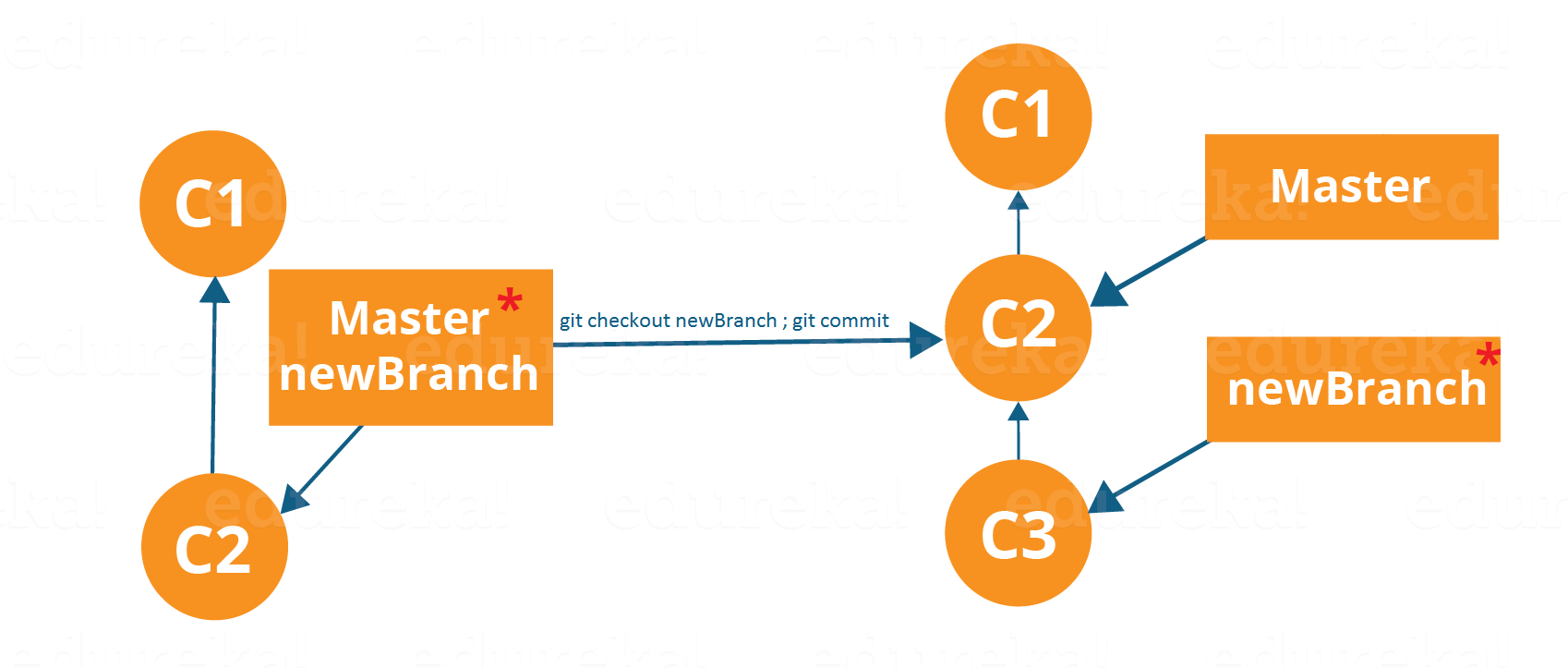
To create a new branch we use the following command:

**git branch <branch-name>**



The diagram above shows the workflow when a new branch is created.  When we create a new branch it originates from the master branch itself.

Since there is no storage/memory overhead with making many branches, it is easier to logically divide up your work rather than have big chunky branches.

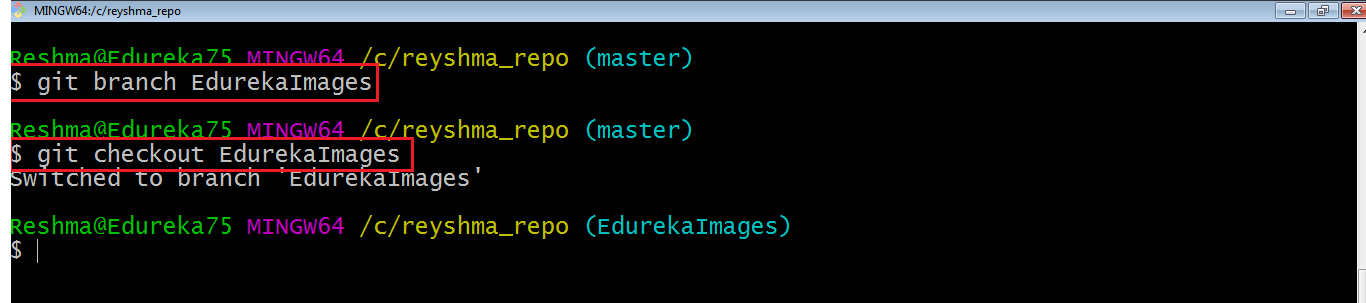
Now, let us see how to commit using branches.

Branching includes the work of a particular commit along with all parent commits. As you can see in the diagram above, the newBranch has detached itself from the master and hence will create a different path.

Use the command below:

**git checkout <branch\_name>**and then

**git commit**



Here, I have created a new branch named “EdurekaImages” and switched on to the new branch using the command **git checkout** .

One shortcut to the above commands is:

**git checkout -b[ branch\_name]**

This command will create a new branch and checkout the new branch at the same time.

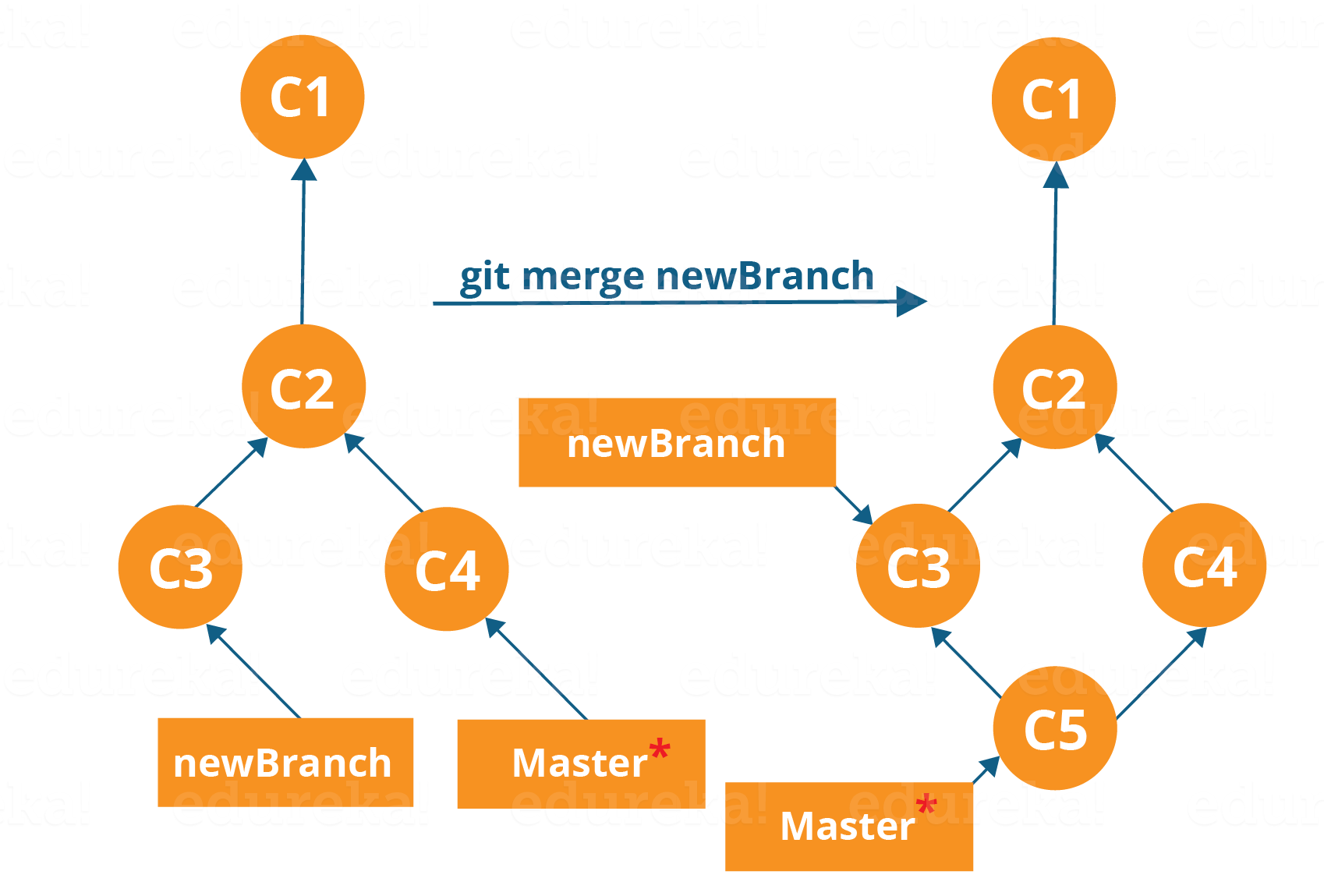
Now while we are in the branch EdurekaImages, add and commit the text file edureka6.txt using the following commands:

**git add edureka6.txt**

**git commit -m”adding edureka6.txt”**

## ****Merging****

Merging is the way to combine the work of different branches together. This will allow us to branch off, develop a new feature, and then combine it back in.



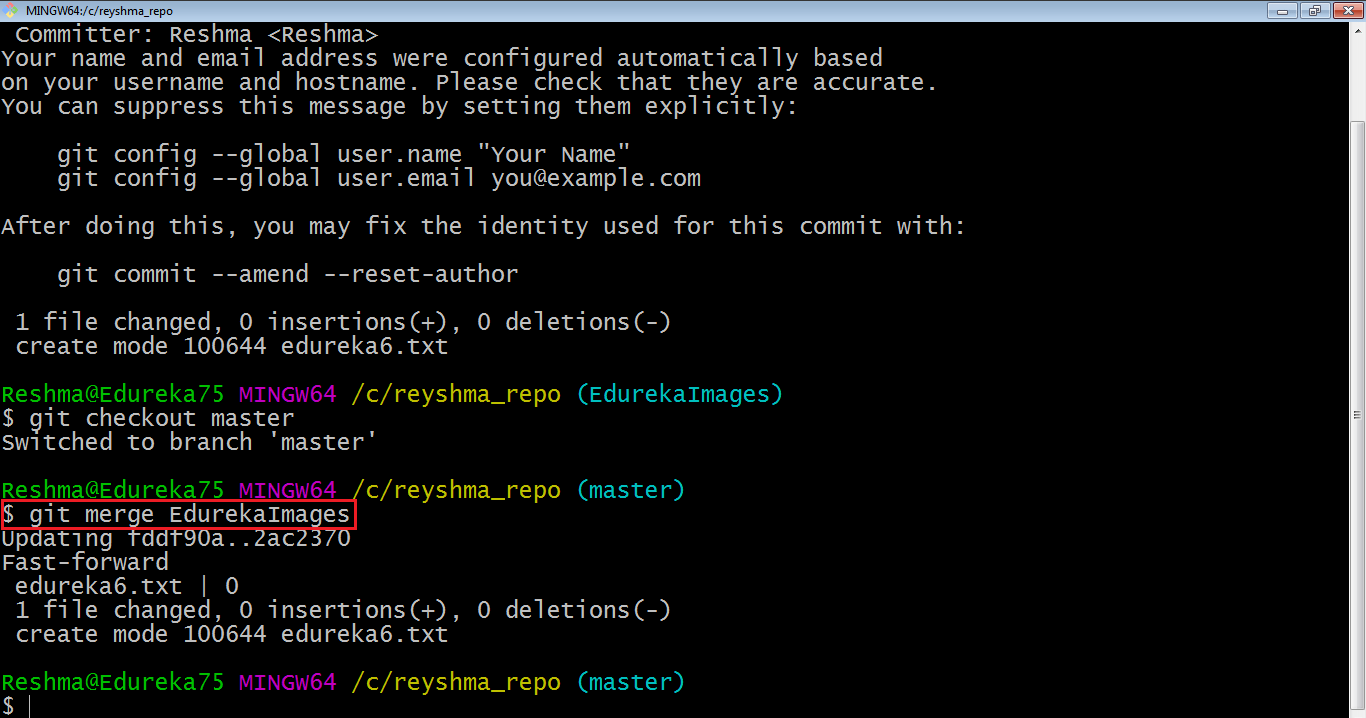
The diagram above shows us two different branches-> newBranch and master. Now, when we merge the work of newBranch into master, it creates a new commit which contains all the work of master and newBranch.

Now let us merge the two branches with the command below:

**git merge <branch\_name>**

It is important to know that the branch name in the above command should be the branch you want to merge into the branch you are currently checking out. So, make sure that you are checked out in the destination branch.

Now, let us merge all of the work of the branch EdurekaImages into the master branch. For that I will first checkout the master branch with the command **git checkout master** and merge EdurekaImages with the command **git merge EdurekaImages**



As you can see above, all the data from the branch name are merged to the master branch. Now, the text file edureka6.txt has been added to the master branch.

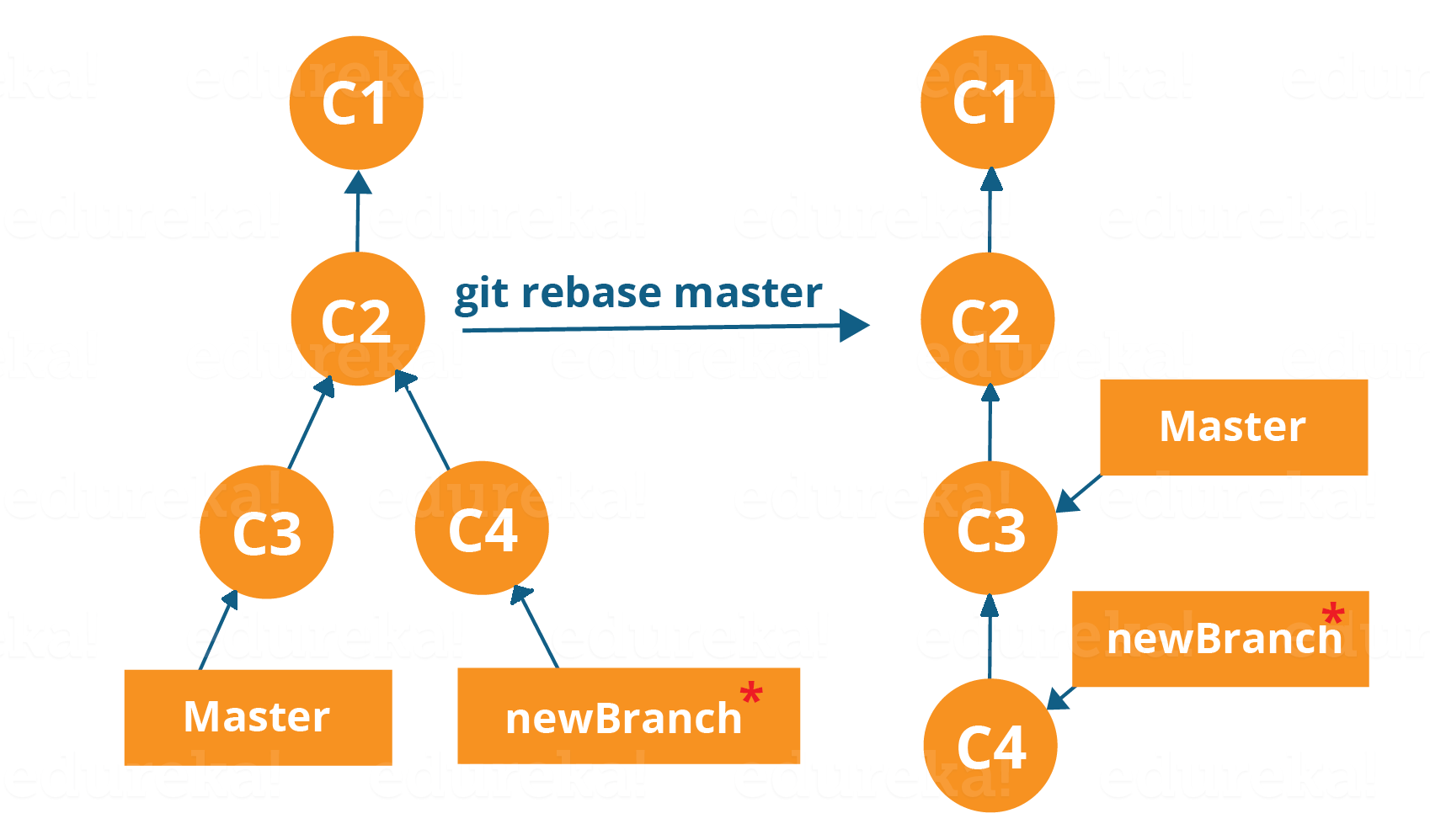
Merging in Git creates a special commit that has two unique parents.

## ****Rebasing****

This is also a way of combining the work between different branches. Rebasing takes a set of commits, copies them and stores them outside your repository.

The advantage of rebasing is that it can be used to make linear sequence of commits. The commit log or history of the repository stays clean if rebasing is done.

Let us see how it happens.

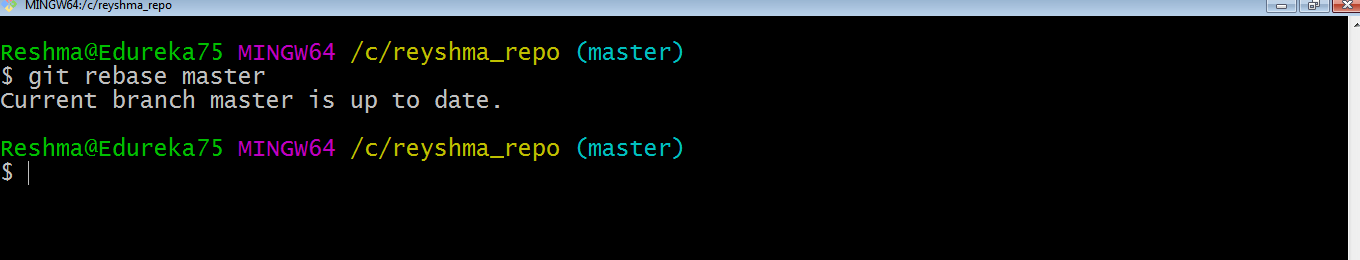


Now, our work from newBranch is placed right after master and we have a nice linear sequence of commits.

**Note**: Rebasing also prevents upstream merges, meaning you cannot place master right after newBranch.

Now, to rebase master, type the command below in your Git Bash:

**git rebase master**



This command will move all our work from current branch to the master. They look like as if they are developed sequentially, but they are developed parallelly.

## ****Git Tutorial – Tips And Tricks****

Now that you have gone through all the operations in this Git Tutorial, here are some tips and tricks you ought to know. :-)

* **Archive your repository**

Use the following command-

**git archive master –format=zip  –output= ../name-of-file.zip**

It stores all files and data in a zip file rather than the **.git** directory.

Note that this creates only a single snapshot omitting version control completely. This comes in handy when you want to send the files to a client for review who doesn’t have Git installed in their computer.

* **Bundle your repository**

It turns a repository into a single file.

Use the following command-

**git bundle create ../repo.bundler master**

This pushes the master branch to a remote branch, only contained in a file instead of a repository.

[[](https://www.edureka.co/devops-certification-training)](https://www.edureka.co/devops-certification-training" \t "_blank)

### [DevOps Certification Training Course](https://www.edureka.co/devops-certification-training" \t "_blank)

[Weekday / Weekend BatchesSee Batch Details](https://www.edureka.co/devops-certification-training" \t "_blank)

An alternate way to do it is:

**cd..**

**git clone repo.bundle repo-copy -b master**

**cd repo-copy**

**git log**

**cd.. /my-git-repo**

* **Stash uncommitted changes**

When we want to undo adding a feature or any kind of added data temporarily, we can “stash” them temporarily.

Use the command below:

**git status**

**git stash**

**git status**

And when you want to re-apply the changes you “stash”ed ,use the command below:

**git stash apply**

I hope you have enjoyed this Git Bash Tutorial and learned the commands and operations in Git. Let me know if you want to know more about Git in the comments section below :-)

[***Git & GitHub certification***](https://www.edureka.co/git-github-sp) has steadily risen from being just a preferred skill to a must-have skill for multiple job roles today. In this blog, I will talk about the Top 20 Git Commands that you will be using frequently while you are working with Git.

Following are the  Git commands which are being covered:

* [**git config**](https://www.edureka.co/blog/git-commands-with-example/#git%20config)
* [**git init**](https://www.edureka.co/blog/git-commands-with-example/#git%20init)
* [**git clone**](https://www.edureka.co/blog/git-commands-with-example/#git%20clone)
* [**git add**](https://www.edureka.co/blog/git-commands-with-example/#git%20add)
* [**git commit**](https://www.edureka.co/blog/git-commands-with-example/#git%20commit)
* [**git diff**](https://www.edureka.co/blog/git-commands-with-example/#git%20diff)
* [**git reset**](https://www.edureka.co/blog/git-commands-with-example/#git%20reset)
* [**git status**](https://www.edureka.co/blog/git-commands-with-example/#git%20status)
* [**git rm**](https://www.edureka.co/blog/git-commands-with-example/#git%20rm)
* [**git log**](https://www.edureka.co/blog/git-commands-with-example/#git%20log)
* [**git show**](https://www.edureka.co/blog/git-commands-with-example/#git%20show)
* [**git tag**](https://www.edureka.co/blog/git-commands-with-example/#git%20tag)
* [**git branch**](https://www.edureka.co/blog/git-commands-with-example/#git%20branch)
* [**git checkout**](https://www.edureka.co/blog/git-commands-with-example/#git%20checkout)
* [**git merge**](https://www.edureka.co/blog/git-commands-with-example/#git%20merge)
* [**git remote**](https://www.edureka.co/blog/git-commands-with-example/#git%20remote)
* [**git push**](https://www.edureka.co/blog/git-commands-with-example/#git%20push)
* [**git pull**](https://www.edureka.co/blog/git-commands-with-example/#git%20pull)
* [**git stash**](https://www.edureka.co/blog/git-commands-with-example/#git%20stash)

So, let’s get started now!!

## ****Git Commands****

### ****git config****

**Usage: git config –global user.name “[name]”**

**Usage: git config –global user.email “[email address]”**

This command sets the author name and email address respectively to be used with your commits.

Git Config Command - Git Commands - Edureka

### ****git init****

**Usage: git init [repository name]**

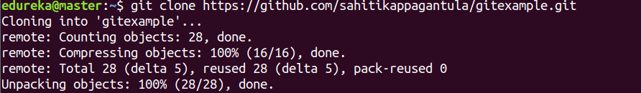
This command is used to start a new repository.

GitInit Command - Git Commands - Edureka

### ****git clone****

**Usage: git clone [url]**

This command is used to obtain a repository from an existing URL.



### ****git add****

**Usage: git add [file]**

This command adds a file to the staging area.

Git Add Command - Git Commands - Edureka

**Usage: git add \***

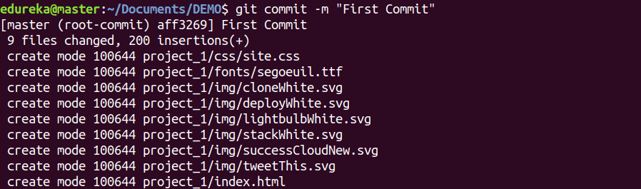
This command adds one or more to the staging area.

Git Add Command - Git Commands - Edureka

### ****git commit****

**Usage: git commit -m “[ Type in the commit message]”**

This command records or snapshots the file permanently in the version history.



**Usage: git commit -a**

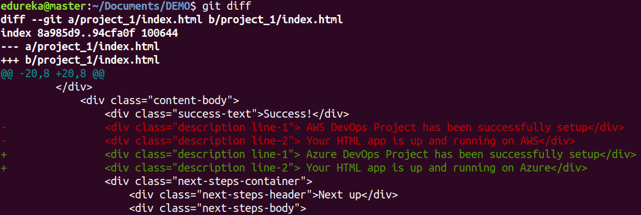
This command commits any files you’ve added with the git add command and also commits any files you’ve changed since then.

Git Commit Command - Git Commands - Edureka

### ****git diff****

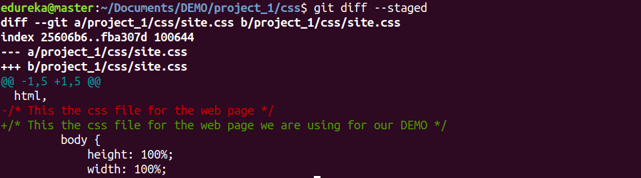
**Usage: git diff**

This command shows the file differences which are not yet staged.



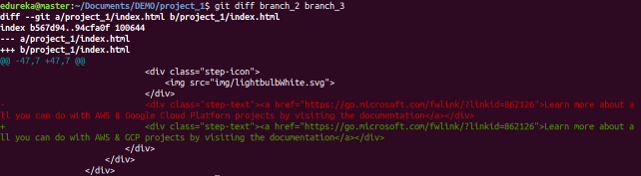
**Usage: git diff –staged**

This command shows the differences between the files in the staging area and the latest version present.



**Usage: git diff [first branch] [second branch]**

This command shows the differences between the two branches mentioned.

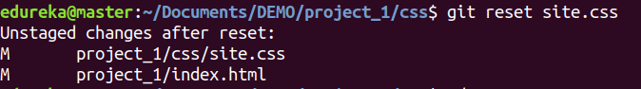


## ****Git Commands With Examples | Edureka****

### ****git reset****

**Usage: git reset [file]**

This command unstages the file, but it preserves the file contents.



**Usage: git reset [commit]**

This command undoes all the commits after the specified commit and preserves the changes locally.

Git Reset Command - Git Commands - Edureka

**Usage: git reset –hard [commit]**

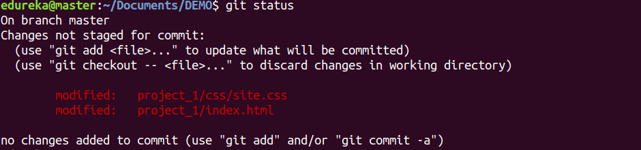
This command discards all history and goes back to the specified commit.

Git Reset Command - Git Commands - Edureka

### ****git status****

**Usage: git status**

This command lists all the files that have to be committed.



### ****git rm****

**Usage: git rm [file]**

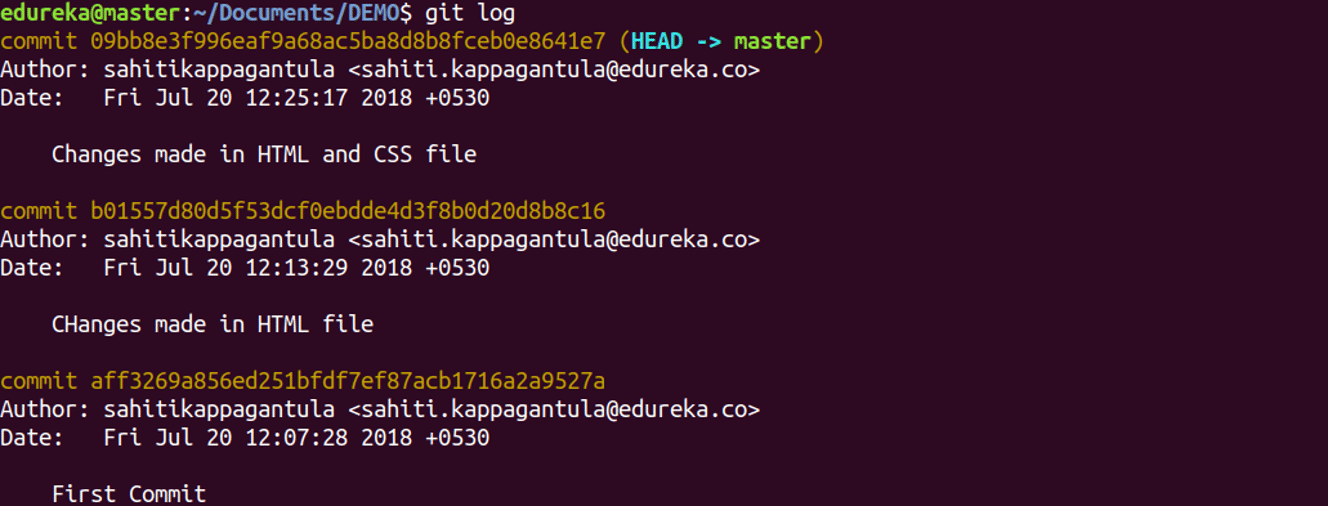
This command deletes the file from your working directory and stages the deletion.

Git Rm Command - Git Commands - Edureka

### ****git log****

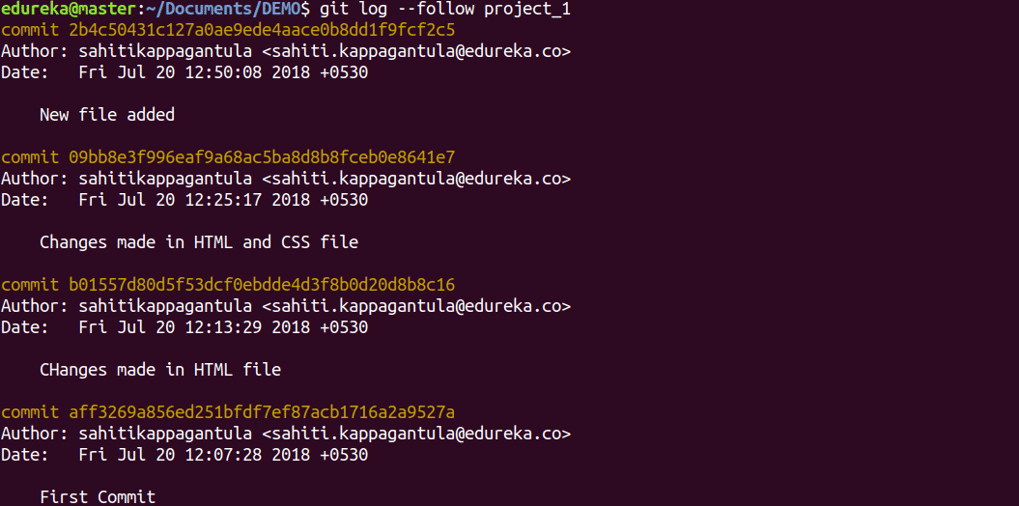
**Usage: git log**

This command is used to list the version history for the current branch.



**Usage: git log –follow[file]**

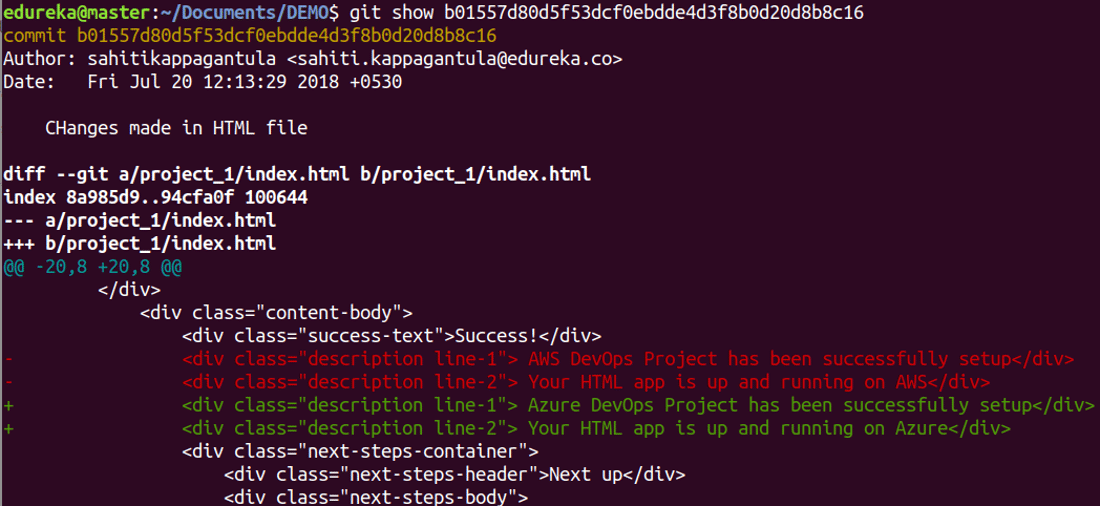
This command lists version history for a file, including the renaming of files also.



### ****git show****

**Usage: git show [commit]**

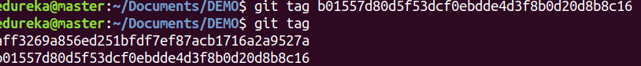
This command shows the metadata and content changes of the specified commit.



### ****git tag****

**Usage: git tag [commitID]**

This command is used to give tags to the specified commit.



### ****git branch****

**Usage: git branch**

This command lists all the local branches in the current repository.

Git Branch Command - Git Commands - Edureka

**Usage: git branch [branch name]**

This command creates a new branch.

Git Branch Command - Git Commands - Edureka

**Usage: git branch -d [branch name]**

This command deletes the feature branch.

Git Branch Command - Git Commands - Edureka

### ****git checkout****

**Usage: git checkout [branch name]**

This command is used to switch from one branch to another.

Git Checkout Command - Git Commands - Edureka

**Usage: git checkout -b [branch name]**

This command creates a new branch and also switches to it.

Git Checkout Command - Git Commands - Edureka

### ****git merge****

**Usage: git merge [branch name]**

This command merges the specified branch’s history into the current branch.

Git Merge Command - Git Commands - Edureka

### ****git remote****

**Usage: git remote add [variable name] [Remote Server Link]**

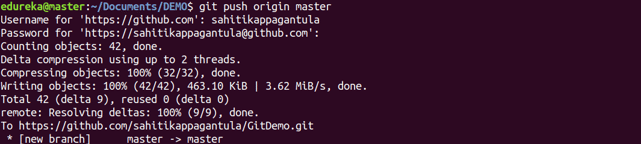
This command is used to connect your local repository to the remote server.

Git Remote Command - Git Commands - Edureka

### ****git push****

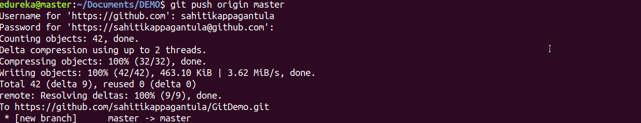
**Usage: git push [variable name] master**

This command sends the committed changes of master branch to your remote repository.



**Usage: git push [variable name] [branch]**

This command sends the branch commits to your remote repository.



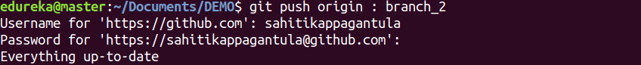
**Usage: git push –all [variable name]**

This command pushes all branches to your remote repository.



**Usage: git push [variable name] :[branch name]**

This command deletes a branch on your remote repository.



### ****git pull****

**Usage:  git pull [Repository Link]**

This command fetches and merges changes on the remote server to your working directory.



### ****git stash****

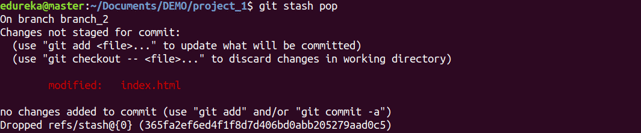
**Usage: git stash save**

This command temporarily stores all the modified tracked files.

Git Stash Command - Git Commands - Edureka

**Usage: git stash pop**

This command restores the most recently stashed files.



**Usage: git stash list**

This command lists all stashed changesets.

Git Stash Command - Git Commands - Edureka

**Usage: git stash drop**

This command discards the most recently stashed changeset.

Git Stash Command - Git Commands - Edureka