Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Beta**

Department: **DCSE**

****

|  |  |  |
| --- | --- | --- |
| **Submitted By:**  Aayush Rai  2110990031  G1 |  | **Submitted To:**  Dr. Monit Kapoor |

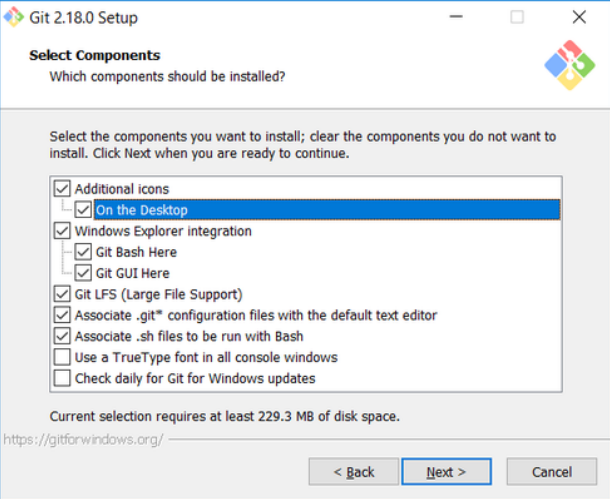
**TASK:1.1 1**

**Aim: Setting up of Git Client Theory:**

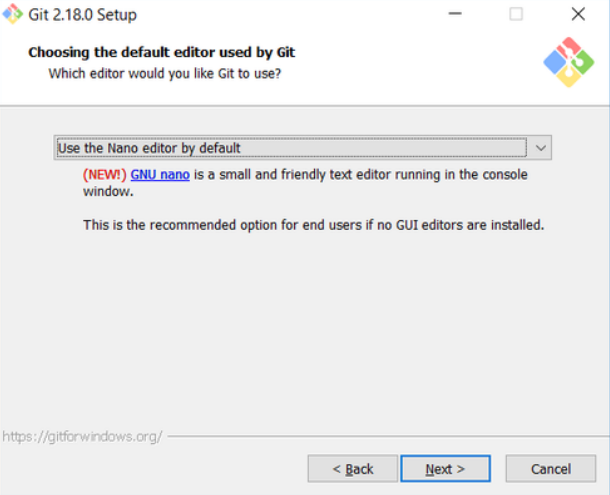
**GIT –> It is basically used for pushing and pulling of code. We can use git and git-hub parallelly to work with multiple members or individually. We can make, edit, recreate, copy or download any code on git hub using git.**

**What is GIT? –> It’s a Version Control System (VCS) -> It is a software, or we can say a server by which we are able to track all the previous changes in the code.**

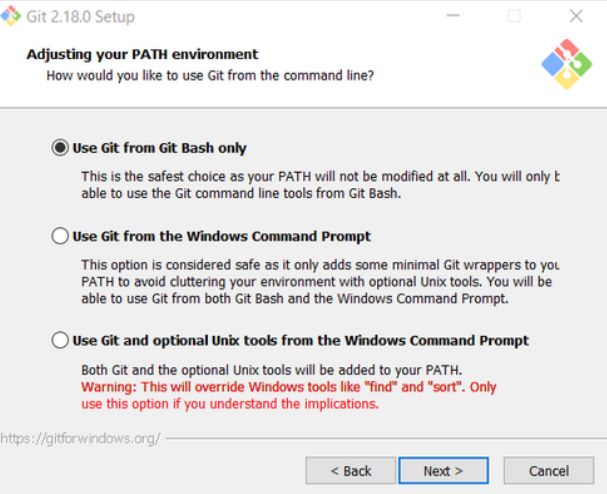
**Advantages of GIT –> Procedure: We can install Git on Windows, using the most official build which is available for download on the GIT’s official website or by just typing (scm-git) on any search engine. We can go on https://git-scm.com/download/win and can select the platform and bit version to download. And after clicking on your desired bit-version or iOS it will start downloading automatically.**

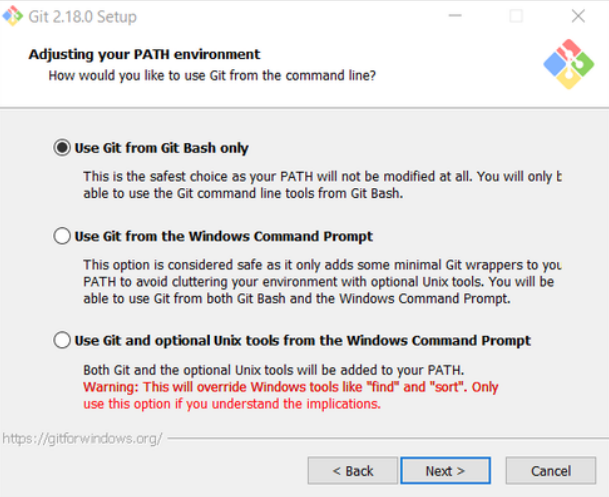


**In the Choosing the default editor used by Git dialog, it is strongly recommended that you DO NOT select the default VIM editor - it is challenging to learn how to use it, and there are better modern editors available. Instead, choose Notepad++ or Nano - either of those is much easier to use. It is strongly recommended that you select Notepad++, BUT YOU MUST INSTALL NOTEPAD++ first! Find the installation with Google.**

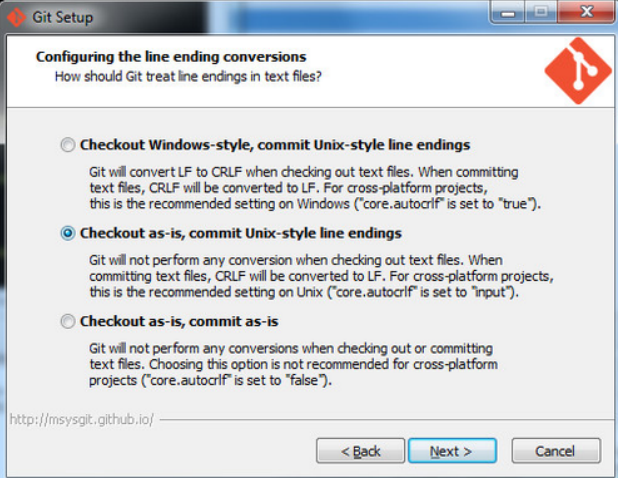
****

**In the Adjusting your PATH screen, all three options are acceptable: Use Git from Git Bash only: no integration, and no extra commands in your command path Use Git from the Windows Command Prompt: adds flexibility - you can simply run git from a Windows command prompt, and is often the setting for people in industry - but this does add some extra commands. Use Git and optional Unix tools from the Windows Command Prompt: this is also a robust choice and useful if you like to use Unix commands like grep.**

****

****

**In the Configuring the line ending screen, select the middle option (Checkout as-is, commit Unix-style line endings) as shown. This helps migrate files towards the Unix-style (LF) terminators that most modern IDE's and editor’s support. The Windows convention (CR-LF line termination) is only important for Notepad (as opposed to Notepad++), but if you are using Notepad to edit your code you may need to ask your instructor for help.**

****

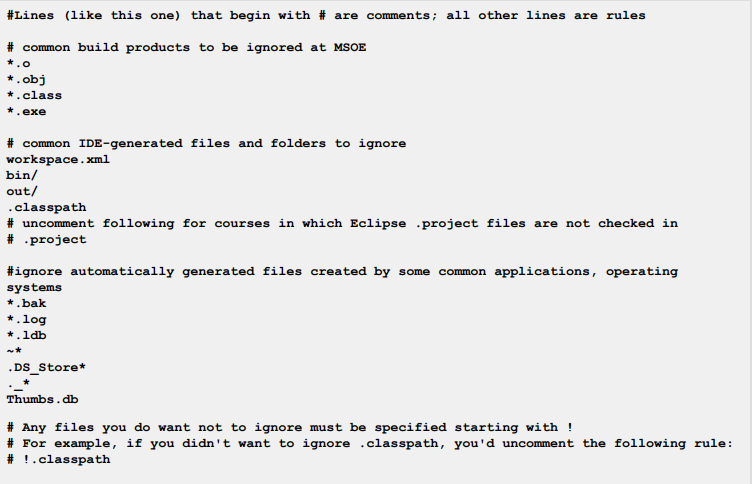
2. Configuring Git to ignore certain files:

**This part is extra important and required so that your repository does not get cluttered with garbage files.**

**By default, Git tracks all files in a project. Typically, this is NOT what you want; rather, you want Git to ignore certain files such as .bak files created by an editor or .class files created by the Java compiler. To have Git automatically ignore particular files, create a file named. gitignore (note that the filename begins with a dot) in the C:\users\name folder (where name is your MSOE login name).**

**NOTE: The. gitignore file must NOT have any file extension (e.g. .txt). Windows normally tries to place a file extension (.txt) on a file you create from File Explorer - and then it (by default) HIDES the file extension. To avoid this, create the file from within a useful editor (e.g., Notepad++ or Ultra Edit) and save the file without a file extension).**

**Edit this file and add the lines below (just copy/paste them from this screen); these are patterns for files to be ignored (taken from examples provided at** [**https://github.com/github/gitignore**](https://github.com/github/gitignore)**.)**

****

**Note: You can always edit this file and add additional patterns for other types of files you might want to ignore. Note that you can also have a. gitignore file in any folder naming additional files to ignore. This is useful for project-specific build products.**

3. Configuring Git default parameters:

**Once Git is installed, there is some remaining custom configuration you must do. Follow the steps below:**

1. **From within File Explorer, right-click on any folder. A context menu appears containing the commands "Git Bash here" and "Git GUI here". These commands permit you to launch either Git client. For now, select Git Bash here.**

**(b)Enter the command (replacing name as appropriate) git config --global core.excludesfile c:/users/name/. gitignore**

* **This tells Git to use the .gitignore file you created in step 2**
* **NOTE: TO avoid typing errors, copy and paste the commands shown here into the Git Bash window, using the arrow keys to edit the red text to match your information.**

**(c)Enter the command git config --global user.email "name@msoe.edu"**

* **This links your Git activity to your email address. Without this, your commits will often show up as "unknown login". Replace name with your own MSOE email name.**

**(d). Enter the command git config --global user.name "Your Name"**

* **Git uses this to log your activity. Replace "Your Name" by your actual first and last name**

**(e) Enter the command git config --global push.default simple**

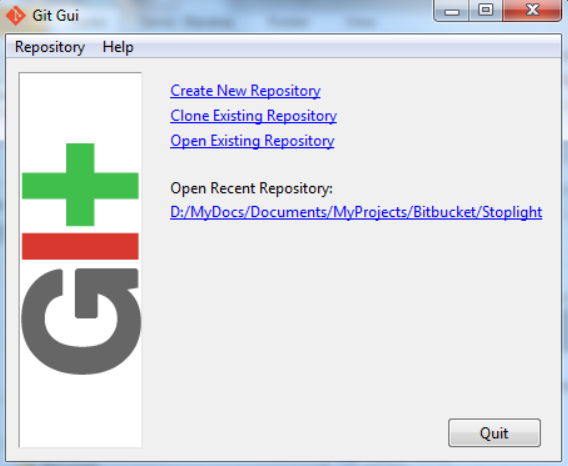
* **This ensures that all pushes go back to the branch from which they were pulled. Otherwise pushes will go to the master branch, forcing a merge.**

4. Generating public/private key pairs for authentication:

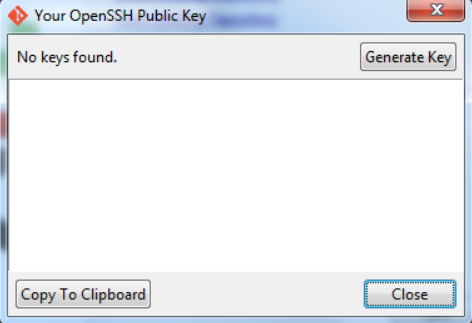
**This part is critical and used to authenticate your access to the repository.**

**You will eventually be storing your project files on a remote Bitbucket or other server using a secure network connection. The remote server requires you to authenticate yourself whenever you communicate with it so that it can be sure it is you, and not someone else trying to steal or corrupt your files. Bitbucket and Git together user public key authentication; thus you have to generate a pair of keys: a public key that you (or your instructor) put on Bitbucket, and a private key you keep to yourself (and guard with your life).**

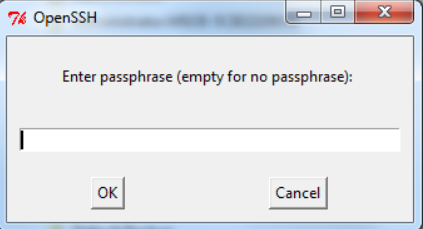
**Generating the key pair is easy: From within File Explorer, right-click on any folder. From the context menu, select Git GUI Here. The following appears:**

****

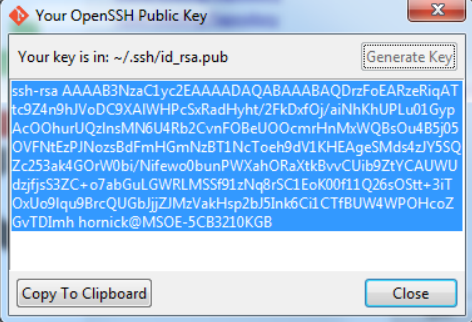
**From the Help menu, select the Show SSH Key command. The following pup-up dialog appears:**

****

**Initially, you have no public/private key pair; thus, the message "No keys found" appears withing the dialog. Press the Generate Key button. The following dialog appears:**

****

**Do NOT enter a passphrase - just press OK twice. When you do, the dialog disappears and you should see something like the following - but your generated key will be different:**

****

**The keys have been written into two files named id\_rsa and id\_rsa.pub in your c:/Users/username/.ssh folder (where username is your MSOE user name). Don't ever delete these files! To configure Bitbucket to use this key:**

* **Click on the Copy to Clipboard button in the Git GUI Public Key dialog.**
* **Log in to Bitbucket.**
* **Click on your picture or the icon in the left pane and select Settings.**
* **Select SSH keys under Security.**
* **Click on the Add key button.**
* **Enter a name for your key in the Label box in the Bitbucket window. If your key is ever compromised (such as someone gets a copy off of your laptop), having a clear name will help you know which key to delete. A good pattern to follow is to name the computer used to generate the key followed by the date you generated it; for instance: "MSOE laptop key 2012-02-28".**
* **2-02-28". Paste the key from the Clipboard into the Key text box in the Bitbucket window, and add it.**

**You should now be able to access your repository from your laptop using the ssh protocol without having to enter a password. Protect the key files - other people can use them to access your repository as well! If you have another computer you use, you can copy the id\_rsa.pub file to the .ssh folder on that computer or (better yet) you can generate another public/private key pair specific to that computer.**

**Configuring other repositories (such as GitLab) is very similar.**

5. Authenticating with private keys

**Linux, Mac users:**

* **Open a terminal prompt.**
* **Type the command.**

**Note that backticks (`) are used, not forward ticks ('). The second command assumes your key is in the default location, ~/.ssh/id\_dsa. If it is somewhere else, type ssh-add path-to-private-key-file. Note that the directory containing the ssh key cannot be readable or writeable by other users; that is, it needs mode 700. You can add these commands to your .bashrc file so they are executed every time you log in. Otherwise the keys only remain active until you close the shell you are using or log out.**

**Windows users:**

* **Install Pageant if it is not installed.**
* **It is usually installed with PuTTY and PuTTYgen.**
* **Start Pageant and select Add Key. Browse to your .ppk file, open it, and enter the passphrase if prompted.**

**If git pull or get push cannot connect, you might need to add a system variable GIT\_SSH set to the path to the plink.exe executable. Go to Windows Settings, enter "system environment" in the search box, open the "Edit the system environment variables" item, click on Environment Variables..., then New... in the System Variables section (the bottom half), enter GIT\_SSH for the name, and browse to plink.exe for the value. Save the setting, then reboot your computer.**

**You will need to re-add the key to Pageant every time you log in to Windows (say, after a reboot). The command start/b pageant c:\path\to\file.ppk will open the file in pageant if you have pageant in your %PATH% variable.**

6. Optional: Configure Git to use a custom application (WinMerge) for comparing file differences:

**It is recommended that you skip this step unless you really are attached to using WinMerge for file comparison tasks.**

**1.Enter the command-**

**This configures Git to use the application WinMerge to resolve merging conflicts. You must have WinMerge installed on your computer first. Get WinMerge at** [**http://winmerge.org/downloads/**](http://winmerge.org/downloads/)**.**

**2.Enter the following commands to complete the WinMerge configuration:**

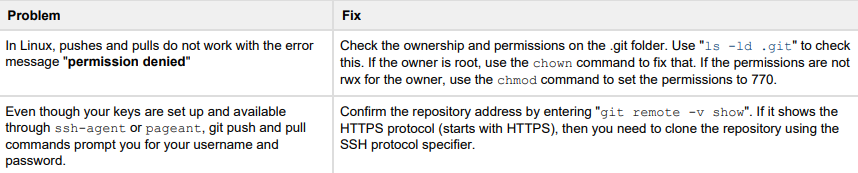
* **git config --global mergetool.winmerge.name WinMerge**
* **git config --global mergetool.winmerge.trustExitCode true.**
* **If you install WinMerge to the default location (that is, C:\Program Files (x86)\WinMerge), enter git config --global mergetool.winmerge.cmd "\"C:\Program Files (x86)\WinMerge\WinMergeU.exe\" -u -e -dl \"Local\" -dr \"Remote\" \$LOCAL \$REMOTE \$MERGED"**
* **If you install WinMerge to an alternate location (for example, D:\WinMerge), enter git config --global mergetool.winmerge.cmd "/d/WinMerge/WinMergeU.exe -u -e -dl \"Local\" -dr \"Remote\" \$LOCAL \$REMOTE \$MERGED".**

**3.Enter the command git config**

* **-This configures Git to use the application WinMerge to differences between versions of files.**

**4.Enter the commands to complete the WinMerge diff configuration:**

**Common problems and possible fixes:**

****

Task No:1.2. 2 

**AIM 2: TO SETUP GITHUB ACCOUNT**

**THEORY:**

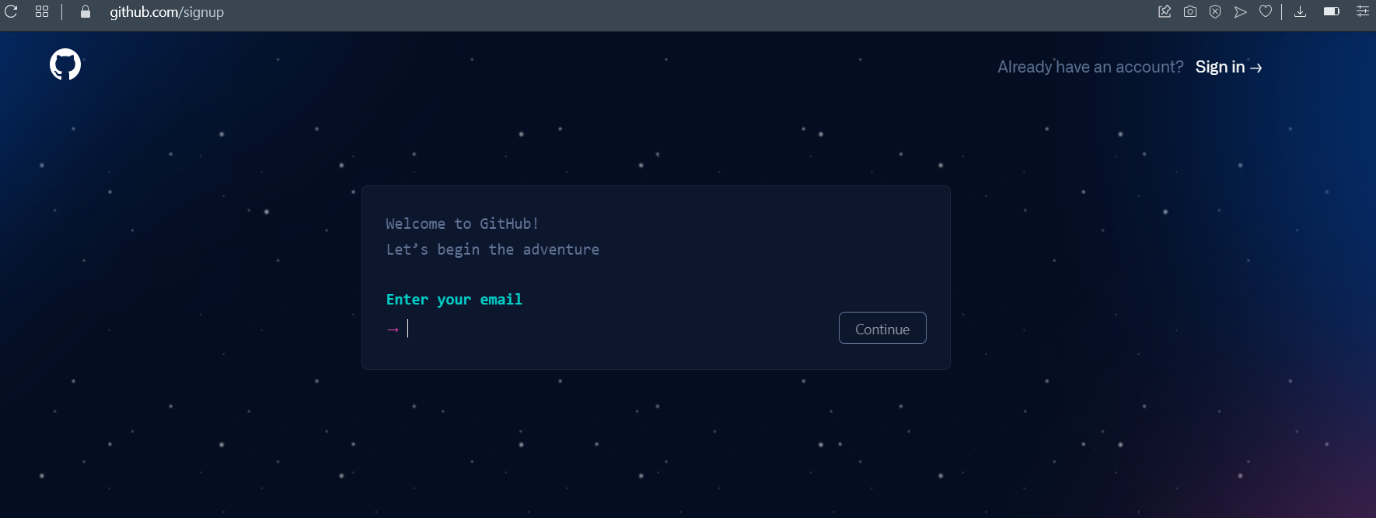
GitHub is a website and cloud-based service (client) that helps an individual or a developer to store and manage their code. We can also track as well as control changes to our or public code. GitHub’s has a user-friendly interface and is easy to use .

We can connect the git-hub and git but using some commands shown below in figure 001. Without GitHub we cannot use Git because it generally requires a host and if we are working for a project, we need to share it will our team members, which can only be done by making a repository . Additionally , anyone can sign up and host a public code repository for free, which makes GitHub especially popular with open-source projects

**PROCEDURE:**

**STEP1:** On any search engine like google, Microsoft edge, opera, etc. , search for git-hub.

A dialogue box would appear welcoming you to the git-hub

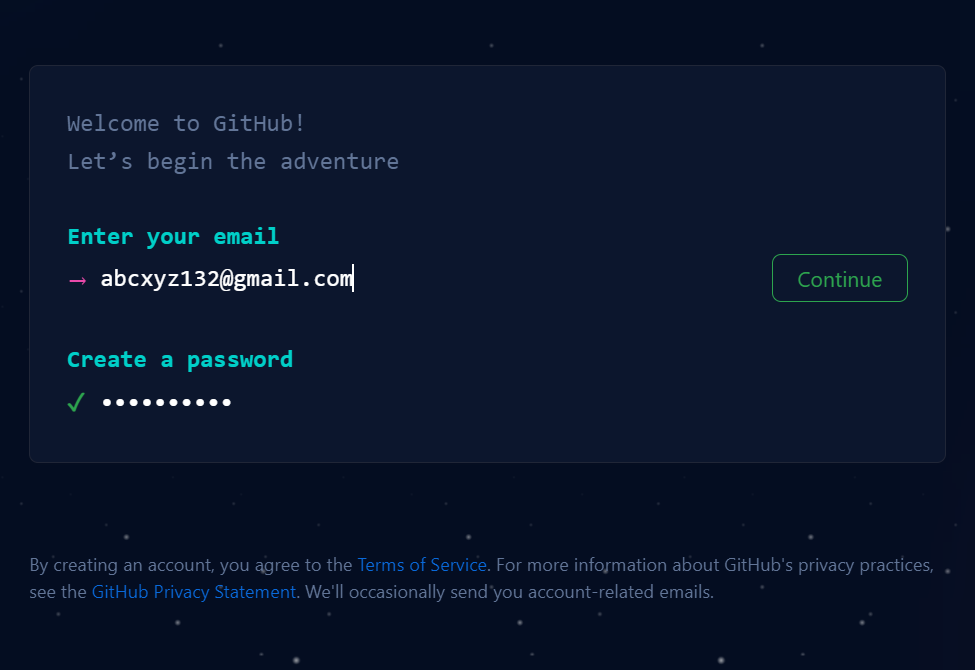


**STEP2: Sign up or log in**

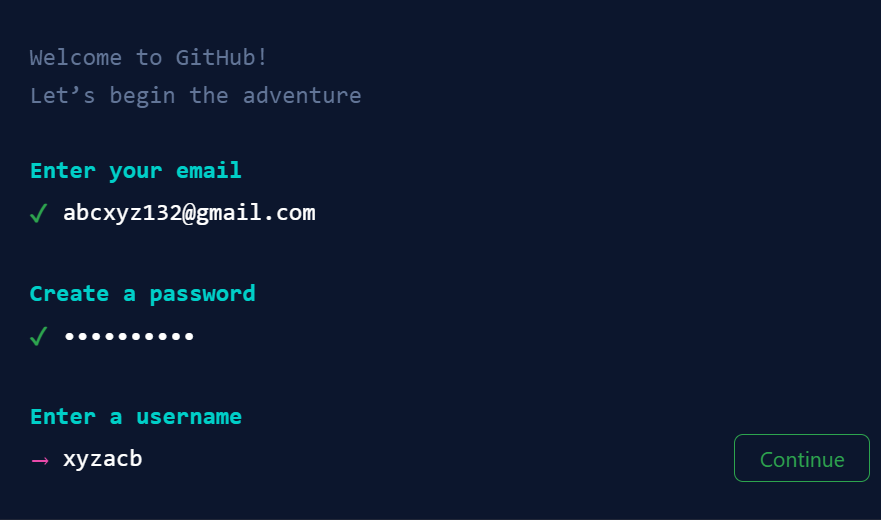
If you already have an account, then on the top right corner there’s an option of signing up. Click on “SIGN IN “.

**STEP 3:** But if you don’t have any account, then enter your email and continue

**STEP 4:** Then create a strong password for your GitHub account



* Create a username.

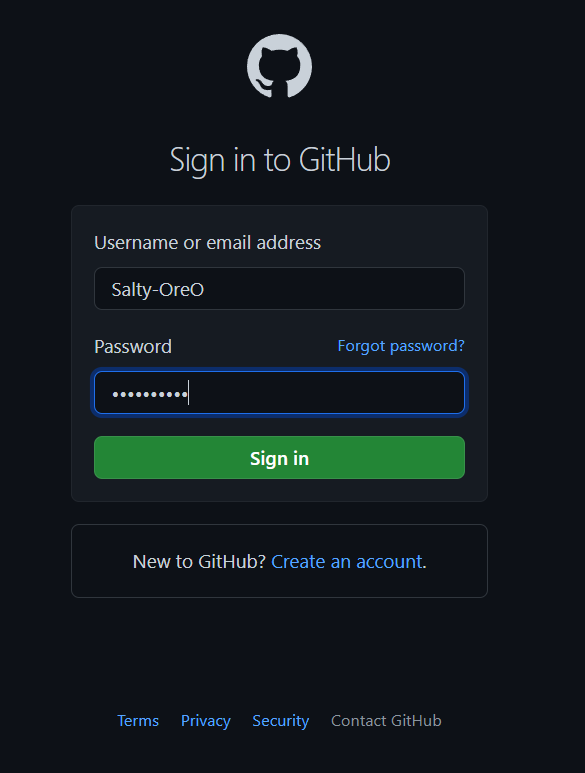
****

* Fill in all the details that are required and click on create account and your GitHub account would be created.

BUT IF YOU ALREADY HAVE AN ACCOUNT, THEN FOLLOW THESE STEPS:

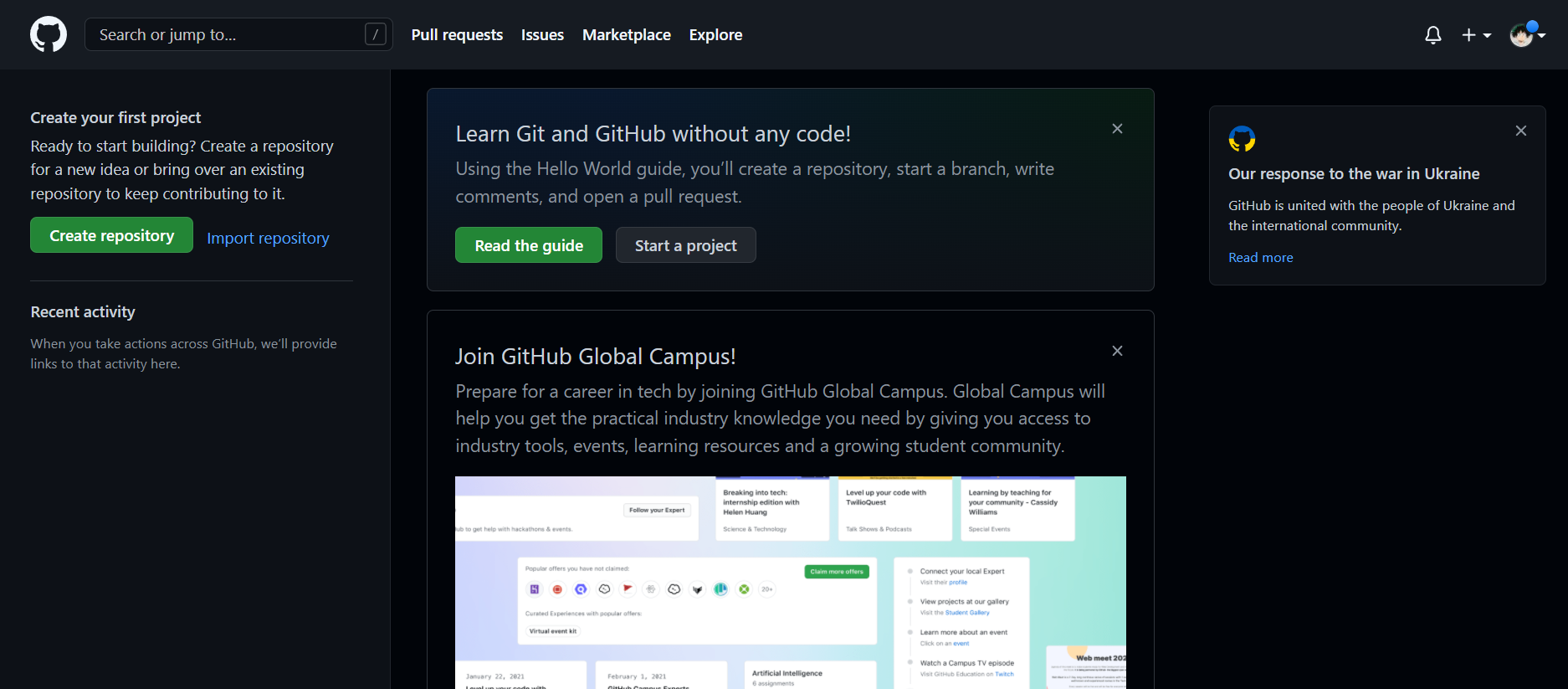
**STEP 1: Sign In**

Enter your email-address or username , then the password and continue.



**STEP 2:** You have logged in in your account. Now you can create and edit any

Project.





**AIM 3: PROGRAM TO GENERATE LOGS**

**THEORY:**

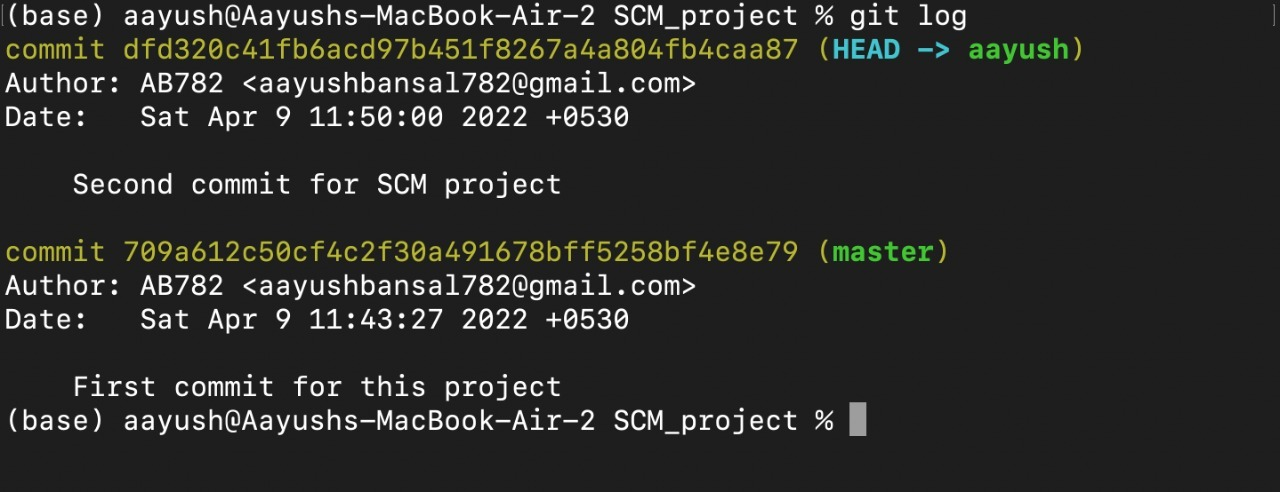
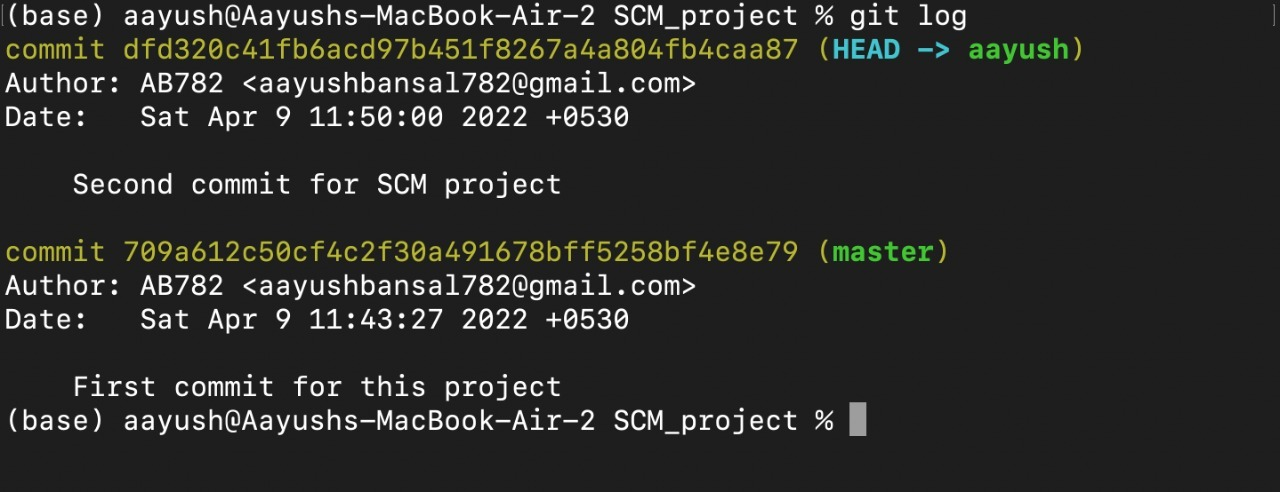
Logs are nothing but the history which we can see in git . It contains all the past commits, insertions and deletions in it which we can see any time.

Logs helps to check that what were the changes in the code or any other file and by whom. It also contains the number of insertions and deletions including at which time it was changed.

**PROCEDURE:**

**The command used to generate logs in git is :**





* As it can be observed, on using this command, the system displays all the changes made in the file or list of all the commits in the history along with the information of the user.
* **This commands clearly defines the git as the ‘version-controlled system’ as it allows us to rollback to any of the previous working states and keeps track of all the versions.**



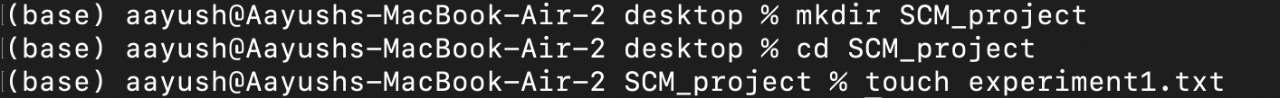
**AIM 4: TO CREATE AND VISUALIZE BRANCHES.**

**THEORY:** The main branch in git is called as master branch. But we can make branches out of this main master branch. All the files present in master can be shown in branch but the file which are created in branch are not shown in master branch. We can also merge both the parent (master) and child (other branches).

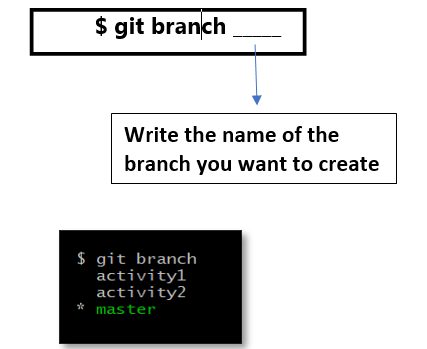
FOLLOW THESE STEPS TO CREATE A SEPARATE BRANCH IN GIT.

1. **Firstly, you can check the present branches in the master branch with the help of this command.**

$ git branch

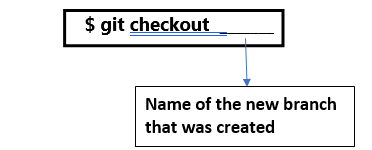


1. **To create a branch, enter:**



* The branch has been created.

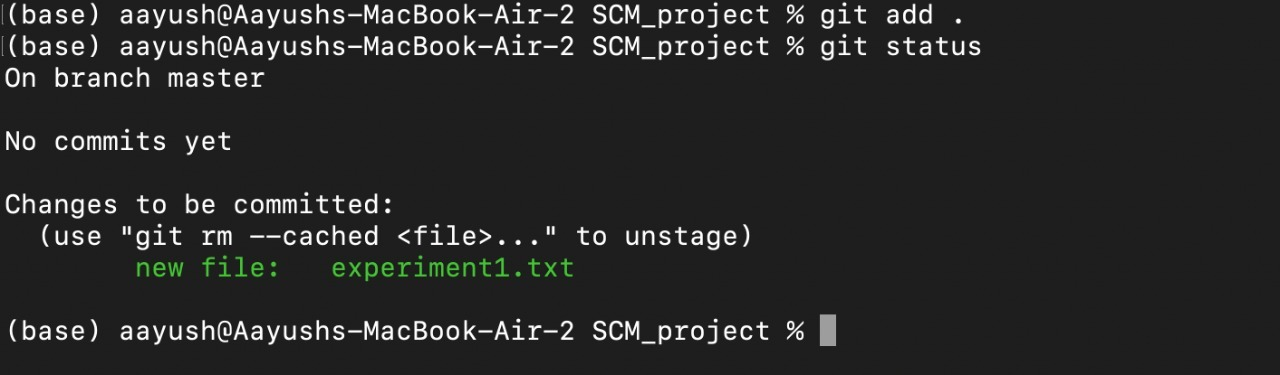
1. **The next step is to transfer the**



**4.Staging the file.**

Check the status of the file.

You would observe the name of the file is in red colour with a notation “untracked “.



This means that only the data has been transferred to the file but we cannot make changes in the same as the current working directory is the master branch.

To overcome this, we stage the file by using the command **:**

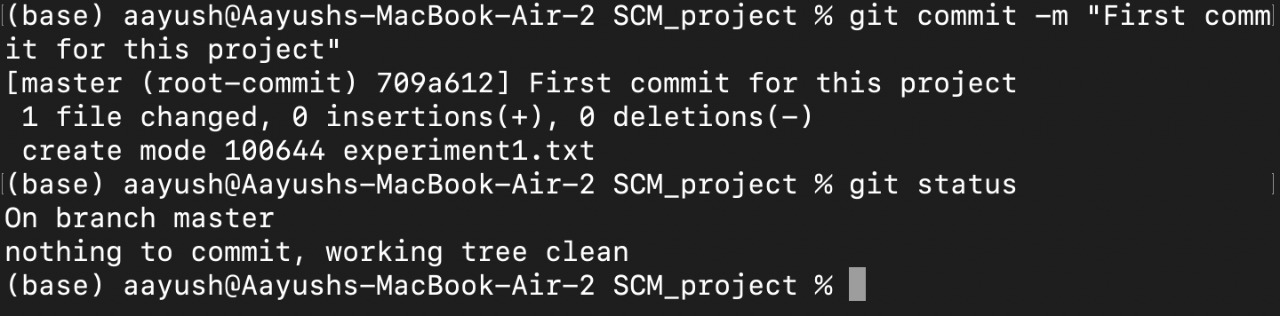
This means that only the data has been transferred to the file but we cannot make changes in the same as the current working directory is the master branch.

To overcome this, we stage the file by using the command **:**

1. **Commit**

After staging we need to commit. This assures the system that the directory has been shifted to the new file. For this, the command used is:

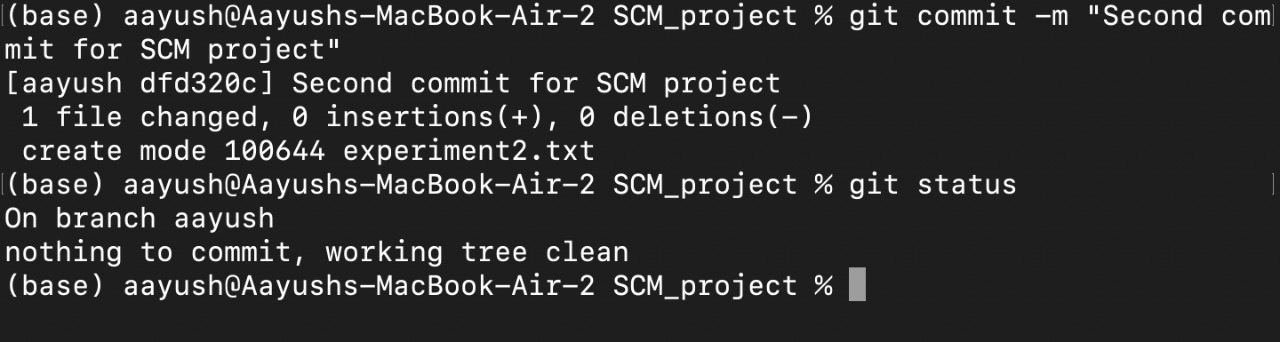
**$ git commit-m \_\_\_\_\_\_\_\_**



1. **Check the status**

On checking the status, a message will be displayed as ‘ working tree clean’. Which means all the files inside that directory are tracked.

Now you can make any of the changes in the file but the modifications won’t be reflected in the master branch.





**AIM5: TO EXPLAIN GIT LIFECYCLE**

**THEORY:** When a project is under Git version control system, they are present in three major Git states in addition to these basic ones. Here are the three Git states**:**

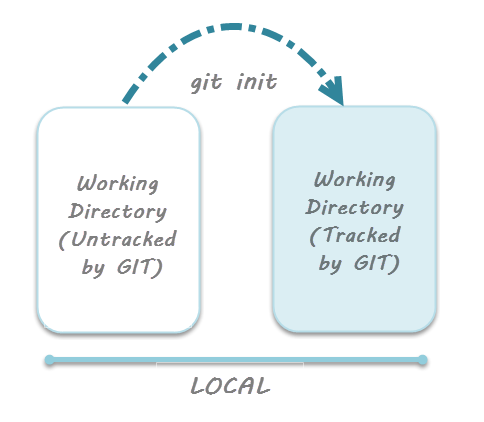
1. **Working directory**
2. **Staging area**
3. **Git directory**

***1.Working Directory***

Consider a project residing in your local system. This project may or may not be tracked by Git. In either case, this project directory is called your Working directory. Working directory is the directory containing hidden .git folder***.***

Working directory is the directory containing hidden .git folder.

**git init** - *Command to initialize a Git repository*



reference for picture: [www.toolsqa.com/git/git-life-cycle/](http://www.toolsqa.com/git/git-life-cycle/)

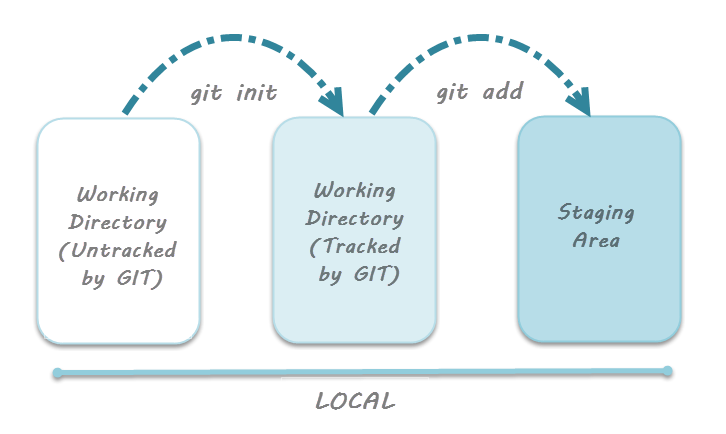
***2.Staging area***

some files in the project like class files, log files, result files and temporary data files are dynamically generated. It doesn't make sense to track the versions of these files.

Whereas the source code files, data files, configuration files and other project artifacts contain the business logic of the application. These files are to be tracked by Git are thus needs to be added to the staging area.

In other words, staging area is the playground where you group, add and organize the files to be committed to Git for tracking their versions**.**

***git add*** - Command to add files to staging area.



{reference for picture: [www.toolsqa.com/git/git-life-cycle/](http://www.toolsqa.com/git/git-life-cycle/) }

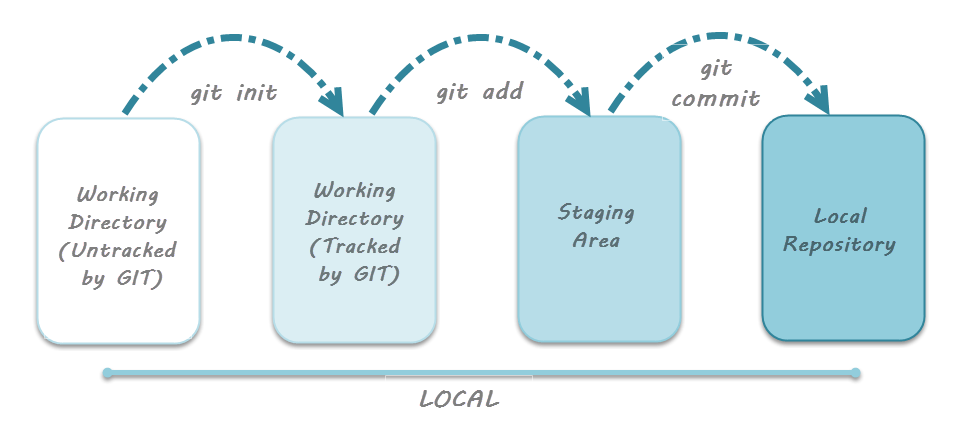
***3.Git Directory***

Now that the files to be committed are grouped and ready in the staging area, we can commit these files. So, we commit this group of files along with a commit message explaining what is the commit about. Apart from commit message, this step also records the author and time of the

commit. Now, a snapshot of the files in the commit is recorded by Git. The information related to this commit is stored in the Git directory.

Thus, Git directory is the database where metadata about project files' history will be tracked.

**git commit -m"your message"** - *Command to commit files to Git repository with message*.



{reference for picture: [www.toolsqa.com/git/git-life-cycle/](http://www.toolsqa.com/git/git-life-cycle/) }