CourseCube
(Formerly Java Learning Center)

DevOps

Module 2

Working with GIT/GitHub
Part 1

Author
Srinivas Dande

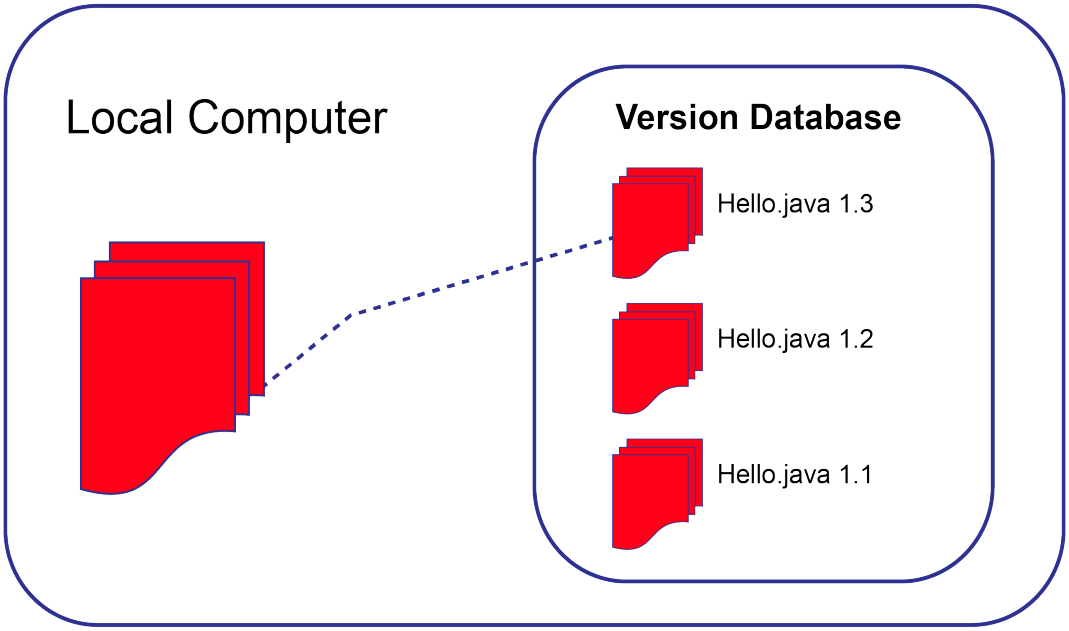
Java Learning Center

1. Version Control System

- Version Control System is software that records changes to a file or set of files over time so that you can recall specific versions later.
- There are 3 types of Version Control Systems
    1) Local Version Control Systems
    2) Centralized Version Control Systems
    3) Distributed Version Control Systems

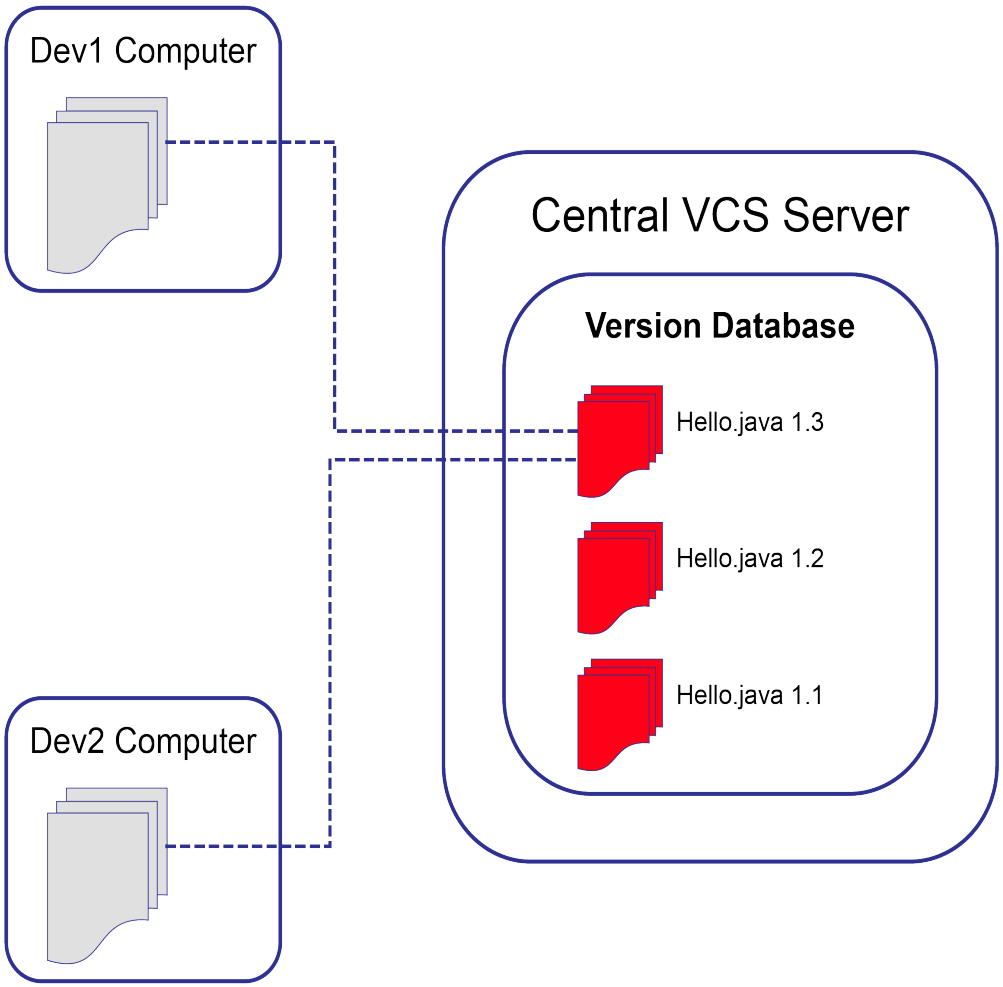
Local Version Control Systems

- A local version control system is a local database located on your local computer, in which every file change is stored as a patch.
- Every patch set contains only the changes made to the file since its last version.
- The main problem with this is that everything is stored locally. If anything happens to the local database, all the patches would be lost.
- Collaborating with other developers or a team is very hard or nearly impossible.



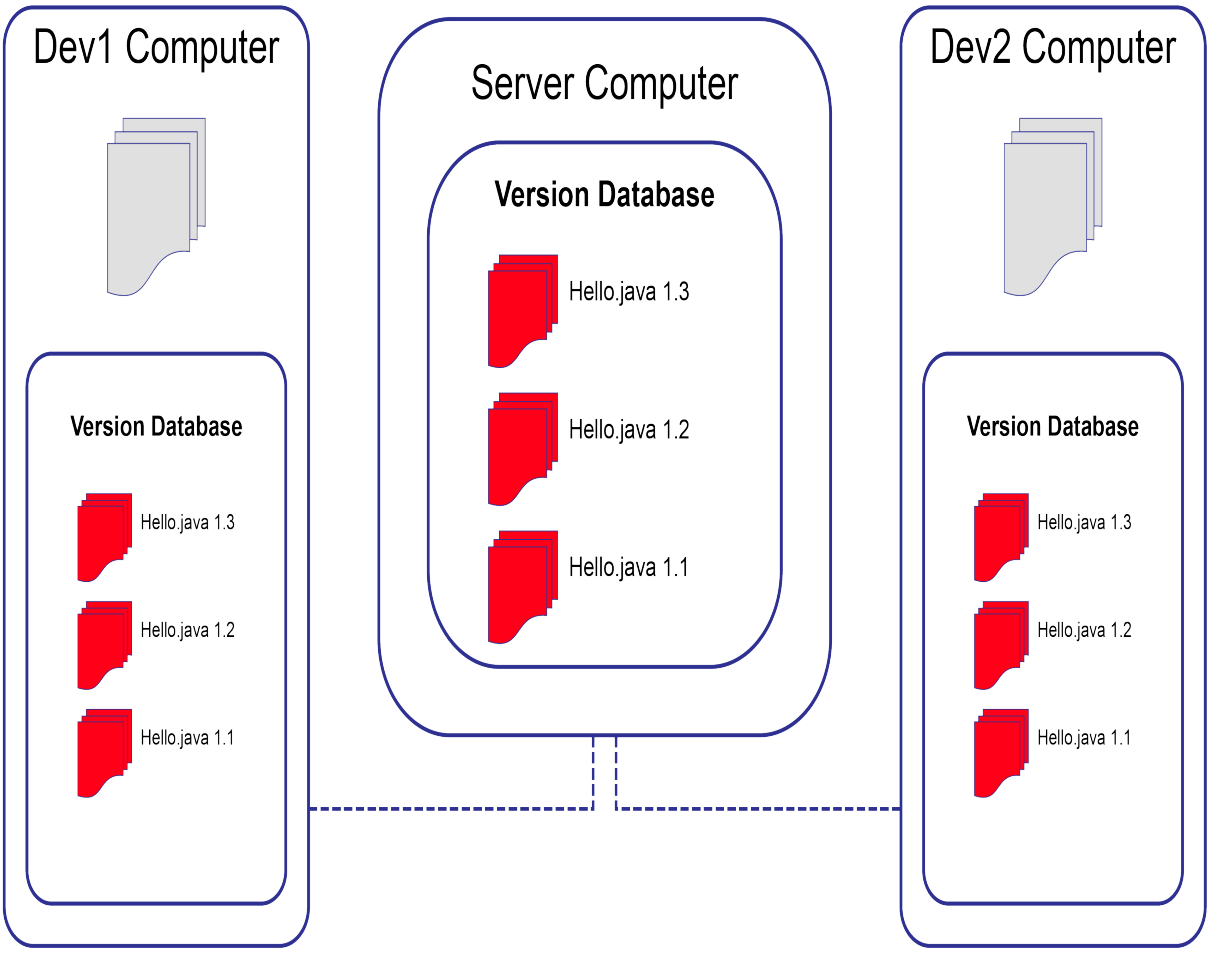
Centralized Version Control Systems

- A centralized version control system has a single server that contains all the file versions.
- This enables multiple clients to simultaneously access files on the server, pull them to their local computer or push them onto the server from their local computer.
- This way, everyone usually knows what everyone else on the project is doing. Administrators have control over who can do what.
- This allows for easy collaboration with other developers or a team.
- The biggest issue with this structure is that everything is stored on the centralized server. If something happens to centralized server, nobody can save their versioned changes,
- CVS, Subversion, and Perforce, ClearCase are Centralized Version Control Systems



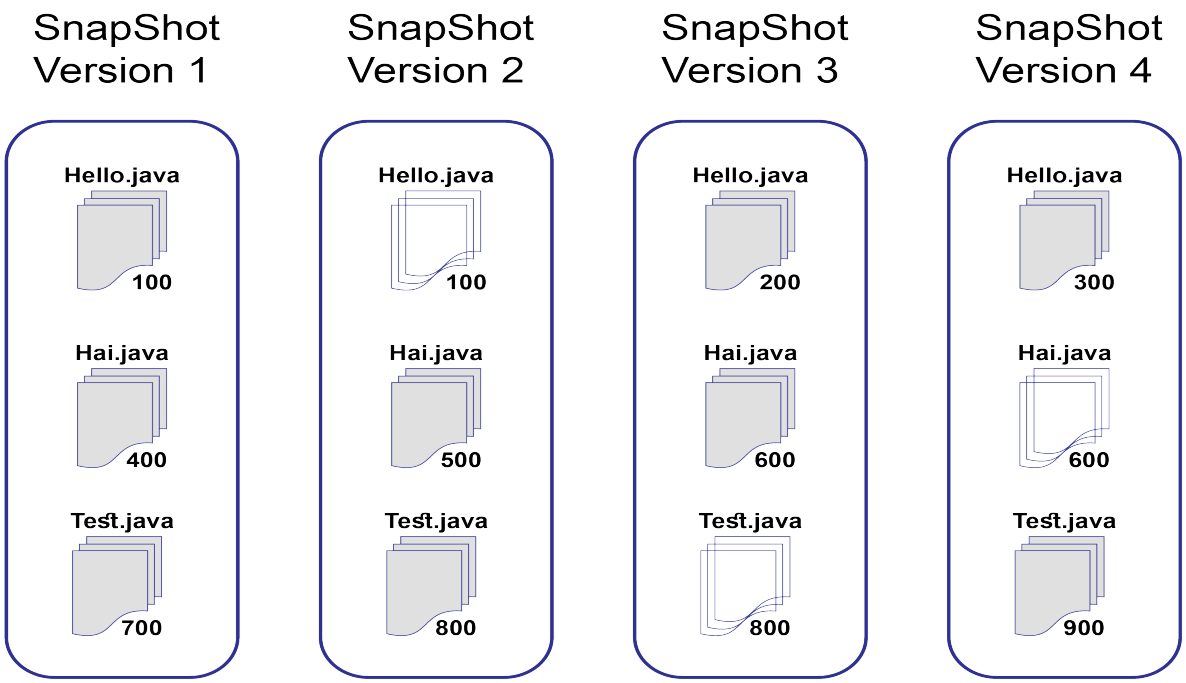
Distributed Version Control Systems

- With distributed version control systems, everyone collaborating on a project owns a local copy of the whole project, i.e. owns their own local database with their own complete history.
- With this model, if the server becomes unavailable or dies, any of the client repositories can send a copy of the project's version to any other client or back onto the server when it becomes available.
- Git is the most popular distributed version control systems.



2. About GIT

- Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
- Every clone is really a full backup of all the data.
- GIT Features:
    1) Speed
    2) Simple design
    3) Strong support for thousands of parallel branches
    4) Fully distributed
    5) Able to handle large projects
- Git thinks about its data as **Stream of Snapshots**
- Whenever you save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot.
- This makes Git more like a mini filesystem with some incredibly powerful tools built on top of it



- **Nearly Every Operation Is Local**
    - Most operations in Git need only local files and resources to operate — generally no information is needed from another computer on your network.
    - Because you have the entire history of the project right there on your local disk, most operations seem almost instantaneous.

3. Three States of GIT

- Git has three main states
    1) Modified
    2) Staged
    3) Committed
- Each file can reside in one of these three states and change states depending on what was done to it.
    - **Modified** - You have changed the file but not yet committed.
    - **Staged** - You have marked a modified file to go into your next commit.
    - **Committed** - Files are safely stored in your local database.
- This leads us to three main sections of a Git project:
    1) Working Tree
    2) Staging Area
    3) Git Repository

Working Tree:

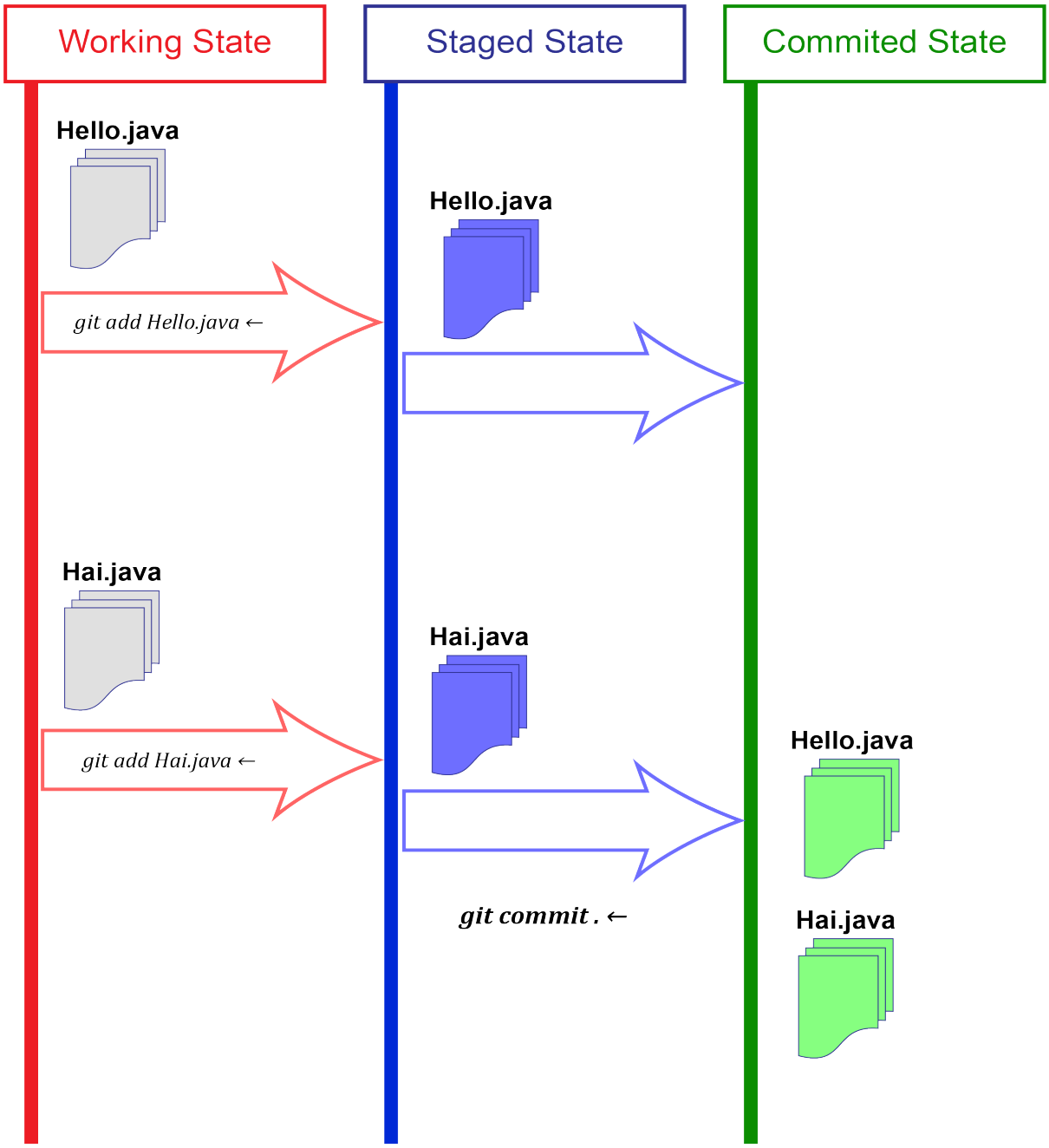
- This is a single checkout of one version of the project.
- This is where you can modify files.

Staging Area:

- It's the area between the working directory and the .git directory.
- All the files which are ready for a commit are stored here.

Git Repository:

- This is the .git directory, also known as the git repository.
- This is where Git stores the metadata and object database for your project.



Git basic workflow:

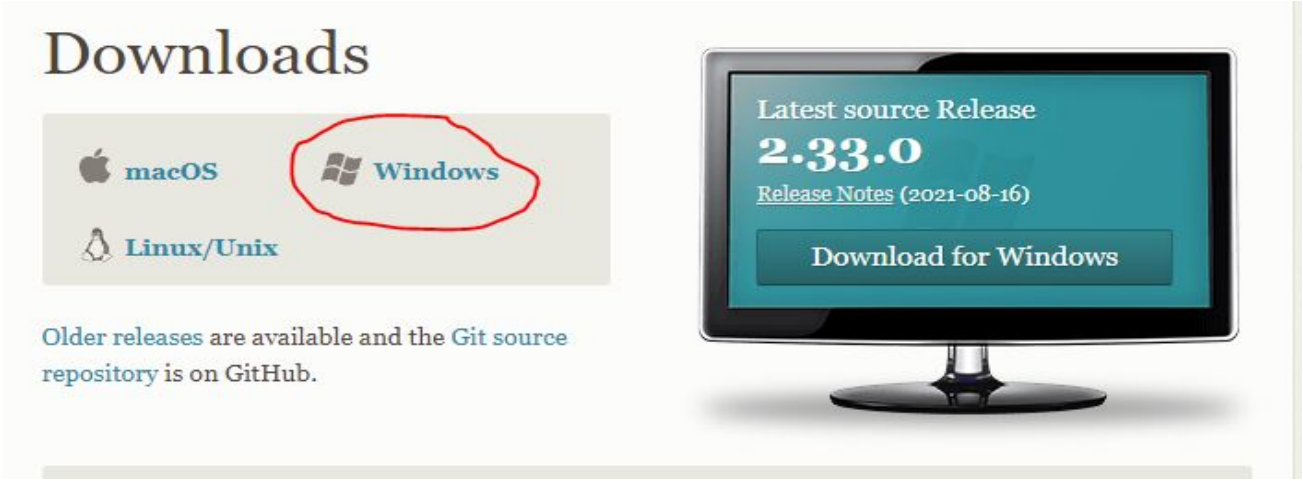
- You modify files in your working tree.
- You selectively stage just those changes you want to be part of your next commit, which adds only those changes to the staging area.
- You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

4. Installing GIT on Ubuntu

$ sudo apt-get install git	Installs GIT in Ubuntu
$ sudo apt-get remove git	Removes GIT from Ubuntu
$ git --version	Tells GIT Version
$ which git	Tells Where GIT Binary is present.

5. Installing GIT on Windows

1) Open https://git-scm.com/downloads
2) You can see the following



3) Click on Windows as marked above
4) That downloads the Installer called - **Git-2.32.0.2-64-bit.**
5) Click on the Installer **- Git-2.32.0.2-64-bit**
6) Provide Installation Location as **E:\GIT** and Follow the Steps to Install with Defaults.
7) Open the Command Prompt and check the Version

git --version	Tells GIT Version

8)

6. GIT Config

- **git config** is a tool that allows you get and set configuration variables.
- These variables can be stored in three different Levels:
  - System Level Variables
  - User Level Variables
  - Repository Level Variables

1) System Level Variables:

- Contains values applied to every user on the system and all their repositories.
- You can use the option **--system** to reads and writes from this file.

Location on Windows: **E:/GIT/etc/gitconfig**

Location on Linux: **/etc/gitconfig**

2) User Level Variables

- Contains values applied to Specific user on the system and Repositories belongs to that Specific User.
- You can use the option **--global** to reads and writes from this file.
- Location:

Location on Windows: **C:/Users/SRINIVAS DANDE/.gitconfig**

Location on Linux: **/home/ubuntu/.gitconfig**

3) Repository Level Variables

- Contains values applied to current working repository only.
- You can use the option **--local** to reads and writes from this file.
- You need to be located somewhere in a Git repository for this option to work properly.
- Location on Windows: **.git/config**
- Location on Linux: **.git/config**

git config --list	Shows GIT Configuration Information

git config --list --show-origin	Shows GIT Configuration Information with Origin

Task 1: Working with git config

In this Task, You will do

1) How to set the Configuration Variables
2) How to get the Configuration Variables
3) How to remove the Configuration Variables

1) See All the Configuration Variables

```
git config --list --show-origin
```

2) Set the Username and Email as System Level Variables

```
sudo git config --system user.name "mysystem"
sudo git config --system user.email "mysystem@jlcindia.com"
```

3) show All the Configuration Variables again

```
git config --list --show-origin
```

4) Show Username and Email Variables

```
git config user.name
git config user.email
```

5) Set the Username and Email as User Level Variables

```
git config --global user.name "myglobal"
git config --global user.email "myglobal@jlcindia.com"
```

6) Show All the Configuration Variables again

```
git config --list --show-origin
```

7) Show Username and Email Variables

```
git config user.name
git config user.email
```

8) Set the Username and Email as Repo Level Variables

```
git config --local user.name "mylocal"
git config --local user.email "mylocal@gmail.com"
```

9) Show All the Configuration Variables again

```
git config --list --show-origin
```

10)Show Username and Email Variables

```
git config --get user.name
git config --get user.email
```

11)You can see the following in **Ubuntu**

```
file:/etc/gitconfig    user.name=mysystem
file:/etc/gitconfig    user.email=mysystem@jlcindia.com

file:/home/ubuntu/.gitconfig   user.name=myglobal
file:/home/ubuntu/.gitconfig   user.email=myglobal@jlcindia.com

file:.git/config       user.name=mylocal
file:.git/config       user.email=mylocal@gmail.com
```

12)You can see the following in **Windows**

file: E:/GIT/etc/gitconfig user.name=mysystem **file: E:/GIT/etc/gitconfig user.email=mysystem@jlcindia.com** **file C:/Users/SRINIVAS DANDE/.gitconfig user.name=myglobal** **file C:/Users/SRINIVAS DANDE/.gitconfig user.email=myglobal@jlcindia.com** **file:.git/config user.name=mylocal** **file:.git/config user.email=mylocal@gmail.com**

Note : user.name and user,email are present in All thre Levels.

13)Show Username and Email Variables

git config --get user.name **git config --get user.email**	Gets user,name and user.email from local

14)Remove the Username and Email as Repo Level Variables

git config --local --unset user.name **git config --local --unset user.email**

15)Show Username and Email Variables

git config --get user.name **git config --get user.email**	Gets user,name and user.email from user

16)Remove the Username and Email as User Level Variables

git config --global --unset user.name **git config --global --unset user.email**

17)Show Username and Email Variables

git config --get user.name **git config --get user.email**	Gets user,name and user.email from system

18)Remove the Username and Email as System Level Variables

sudo git config --system --unset user.name **sudo git config --system --unset user.email**

19)Show Username and Email Variables

git config --get user.name **git config --get user.email**	Nothing will be shown

Note: Use Global Variables Always

20)Set the Username and Email as User Level Variables

git config --global user.name "DandesClasses" **git config --global user.email "dandesclasses@gmail.com"**

7. Basic GIT Workflow

Task 2: Basic GIT Workflow

In this Task, You will do

1) Initialzing GIT Repository
2) Add New Files
3) Add Files to Stage
4) Commit the Files
5) Update the Existing Files
6) Add Files to Stage
7) Commit the Files
8) See the Logs

1) Initialize the GIT Repository

```
mkdir myjlc-repo
cd myjlc-repo
git init
git status
```

2) Create the New File and check the Status

```
touch Hello.java
git status

echo "# myjlc-repo- My First GIT Repo" >> README.md
git status
```

3) Add the Files to Stage

```
git add Hello.java
git status

git add README.md
git status
```

4) Commit the Files

```
git commit -m "my first commit"
git status
```

5) Update Existing Files

```
vim Hello.java                    -> Modify Hello.java in VI Editor
git status
```

6) Add the Files to Stage

```
git add Hello.java
git status
```

7) Commit the Files

```
git commit -m "my second commit"
git status
```

8) See GIT Logs

```
git log
git log --oneline
```

8. Removing files from Untracked/Unstaged State

- When You Create the New File or When You Modify the Existing File , Then That File will be placed in **Untracked State**
- You can remove File from **Untracked State** using **git clean**
- Below are the List of Options for **git clean**

git clean -n	- To see which files will be deleted
git clean -f	- To delete the files
git clean -f <fname>	- To delete the specified file
git clean -f -d	- To remove directories
git clean -fd	- To remove directories
git clean -f -X	- To remove ignored files
git clean -fX	- To remove ignored files
git clean -f -x	- To remove ignored and non-ignored files
git clean -fx	- To remove ignored and non-ignored files

Task 3: Removing Files from Untracked State

In this Task, You will do

1) Add New Files
2) Remove Seletive Files from Untracked State
3) Remove All Files from Untracked State

1) Create the New File and check the Status

```
touch test1.java
touch test2.java
touch test3.java
touch test4.java
touch test5.java
git status
```

2) Add the Files to Stage

```
git add test1.java
git status
```

3) Commit the Files

```
git commit -m "my commit - 3"
git status
```

4) Add the Files to Stage

```
git add test2.java
git status
```

5) Check Which files will be deleted from **Untracked State**

```
git clean -n
```

6) Remving test3.java from **Untracked State**

```
git clean -f  test3.java
git status
```

7) Remving All the files from **Untracked State**

```
git clean -f
git status
```

8) See GIT Logs

```
git log
```

9. Removing files from Staged State

- When You run **git add**, Then the File will be moved in **Staged State**
- You can remove File from **Staged State** using the following commnds

  git rm --cached <filename>

  git rm --force <filename>

  git reset <filename>

  git reset

Task 4: Removing Files from Staged State

In this Task, You will do

1) Add New Files
2) Add the Files to Staged State
3) Remove the Files from Staged State

1) Create the New File and check the Status

```
echo "I am test1.java">> test1.java
git status
```

2) Add the Files to Stage

```
git add test1.java
git status
```

3) Move the Files from Staged to Untracked State

```
git rm --cached test1.java
git status
```

4) Add the Files to Stage

```
git add test1.java
git status
```

5) Remving test1.java from Staged State completely

```
git rm --force test1.java
git status
```

6) Create the New File and check the Status

```
echo "I am test1.java">> test1.java
git status
```

7) Add the Files to Stage

```
git add test2.java
git status
```

8) Move the Files from Staged to Untracked State

```
git reset test1.java
git status
```

9) Create the New File and check the Status

```
echo "I am test2.java">> test2.java
git status
```

10)Add the Files to Stage

```
git add test2.java
git status
```

11)Move the Files from Staged to Untracked State

```
git reset
git status
```

10. Removing files from Commited State

- When You run **git commit**, Then the File will be moved in **Commited State**
- You can remove File from **Commited State** using the following commnds

  git rm --cached <filename>

  git rm --force <filename>

  git reset <filename>

  git reset

Task 5: Removing Files from Staged State

In this Task, You will do

1) Add New Files
2) Add the Files to Staged State
3) Remove the Files from Staged State

1) Create the New File and check the Status

```
echo "I am Hello.java">> Hello.java
git status
```

2) Add the Files to Stage

```
git add Hello.java
git status
```

3) Commit the Files

```
git commit -m "my commit- 4 "
git status
```

4) Move the File from Commited State to Staged State

```
git rm Hello.java
git status
```

5) Moves from Staged state to Untracked Stsate

```
git restore --staged Hello.java
git status
```

6) ReStore the File (Moves from Untracked Stsate to Commited Stsate)

```
git restore Hello.java
git status
```

7) Delete Permenently

```
git add Hello.java
git commit -m "deleting Hello.java"
```

11. Moving Files from One Folder to another

- You can move the Files from One Folder to another using **git mv** commnd

  git mv <Path>/<FileName> <Path>/<FileName>

Task 6: Moving Files /Rename Files

In this Task, You will do

1) Create the Folder
2) Move File to Another Folder
3) Rename the File

1) Create the Folder called myjlc

```
mkdir myjlc
```

2) Move the File to myjlc folder

```
git mv Hello.java myjlc
git status
```

3) Commit the Move Operation

```
git commit  -m "Moving  Hello.java"
```

4) Rename test2.java to test1.java

```
git mv test2.java test1.java
git status
```

5) Commit the Rename Operation

```
git commit  -m "Rename to test1.java"
```

12. Rollback the Commits

- You can Rollback the Commits at any time with **git reset** commangd

  git reset –soft <CommitId>

  git reset –hard <CommitId>

Task 7: Rollback the Commits

In this Task, You will do

1) See the Commits Happed till now
2) Select the Commit ID which you want to rollback.
3) Rollback the Commit

1) See the Commits

```
git log --oneline
```

2) Do soft Reset

```
git reset --soft 7e234c6
```

3) Do hard Reset

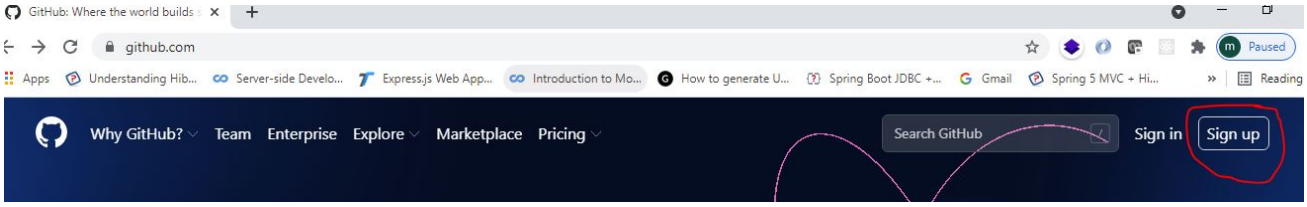
```
git reset --soft 7e234c6
```

4) See the Commits

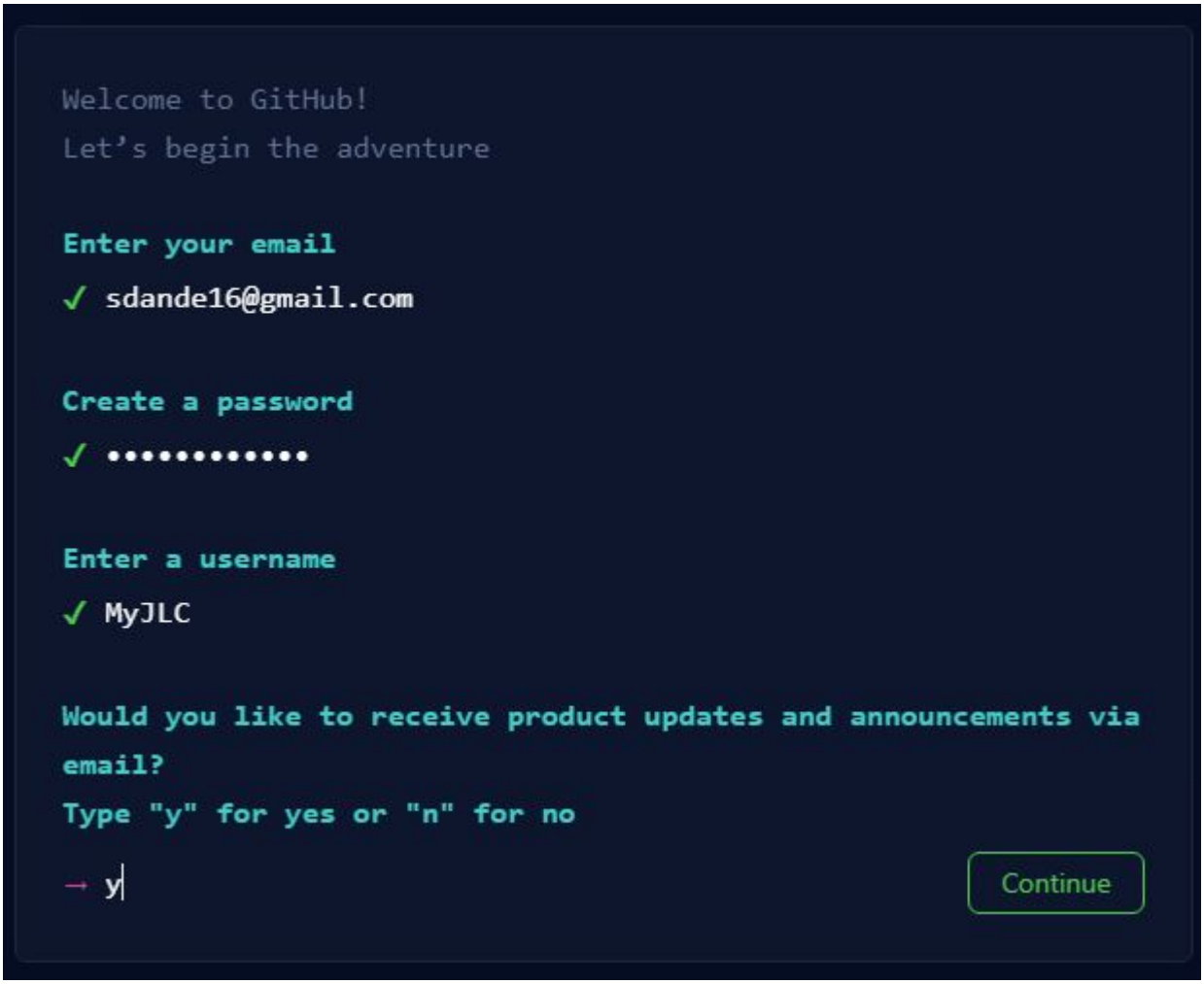
```
git log --oneline
```

13. Setup GitHub Account

- Create the Account in GitHub with the following Steps
- Open https://github.com and click on Signup.

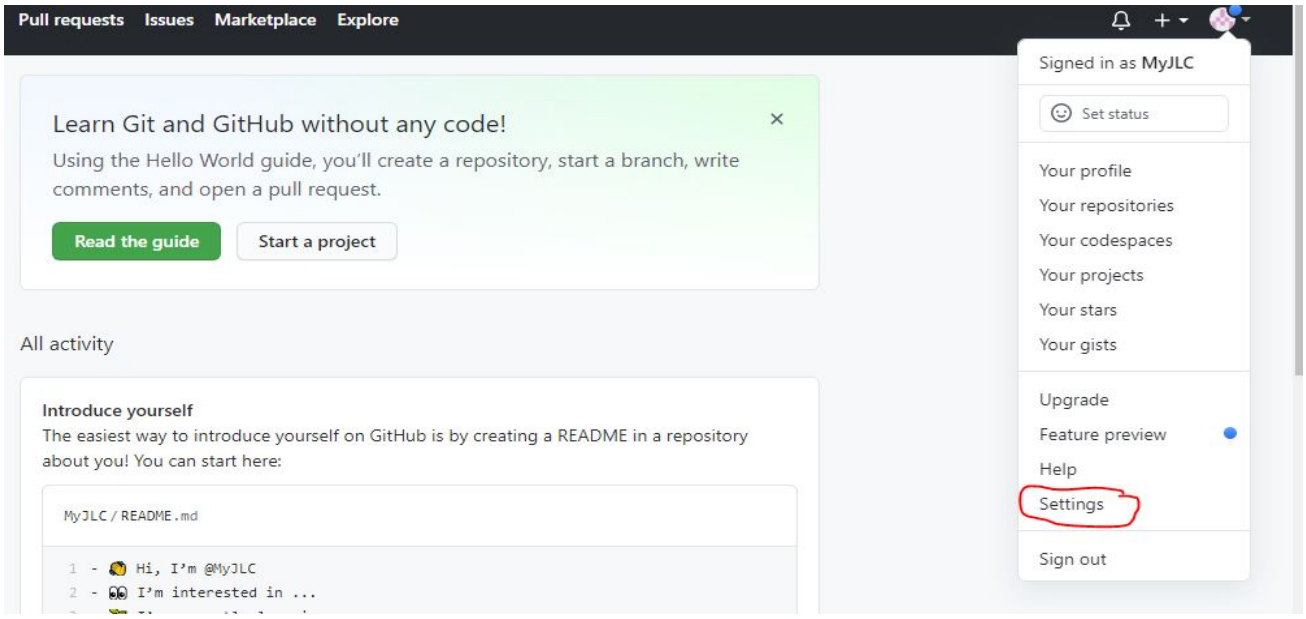


- Follow the Steps given One by one.

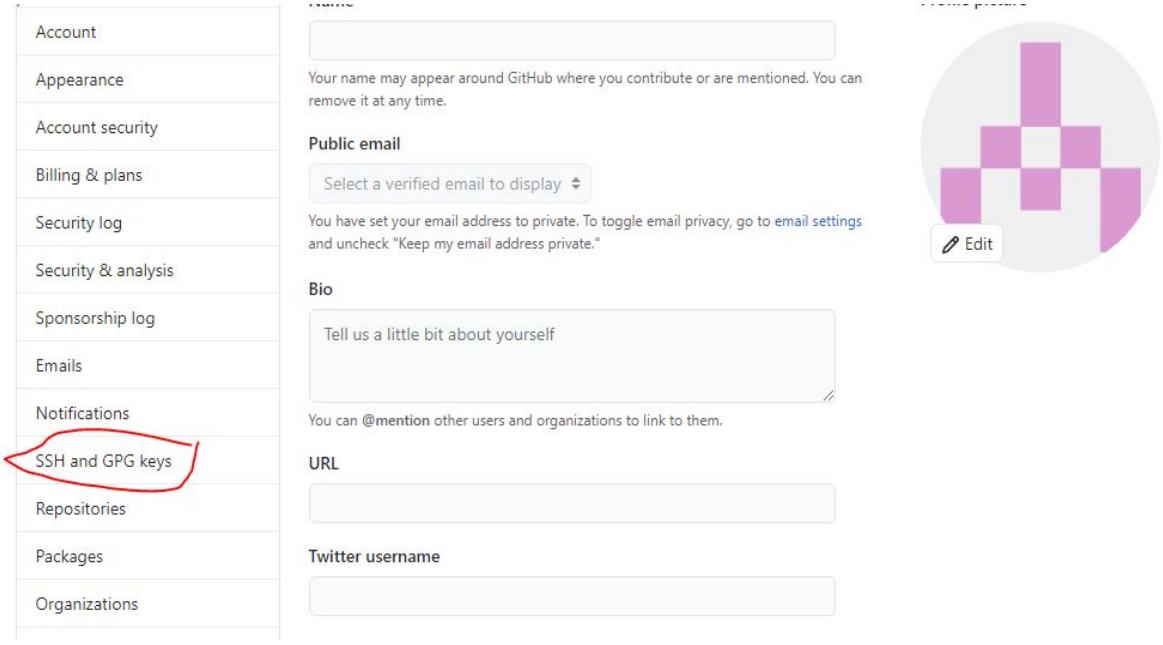


14. Add SSH Key to your GitHub

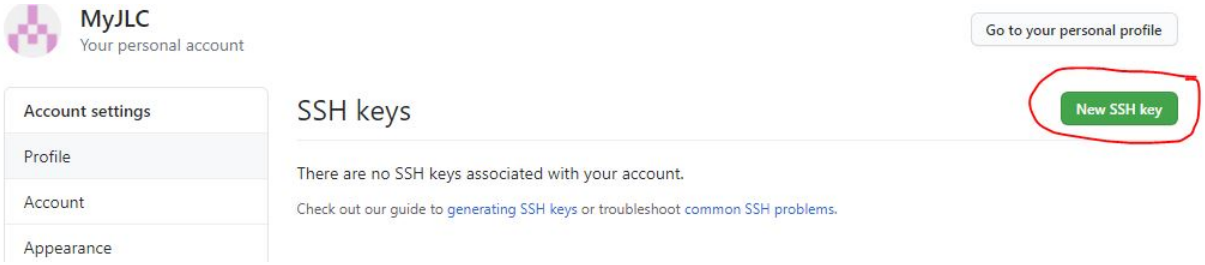
1) Generate public and private keys
    - **ssh-keygen** (same in Windows and Linux)
2) Login to Github Account.
3) **Select Settings as show below.**



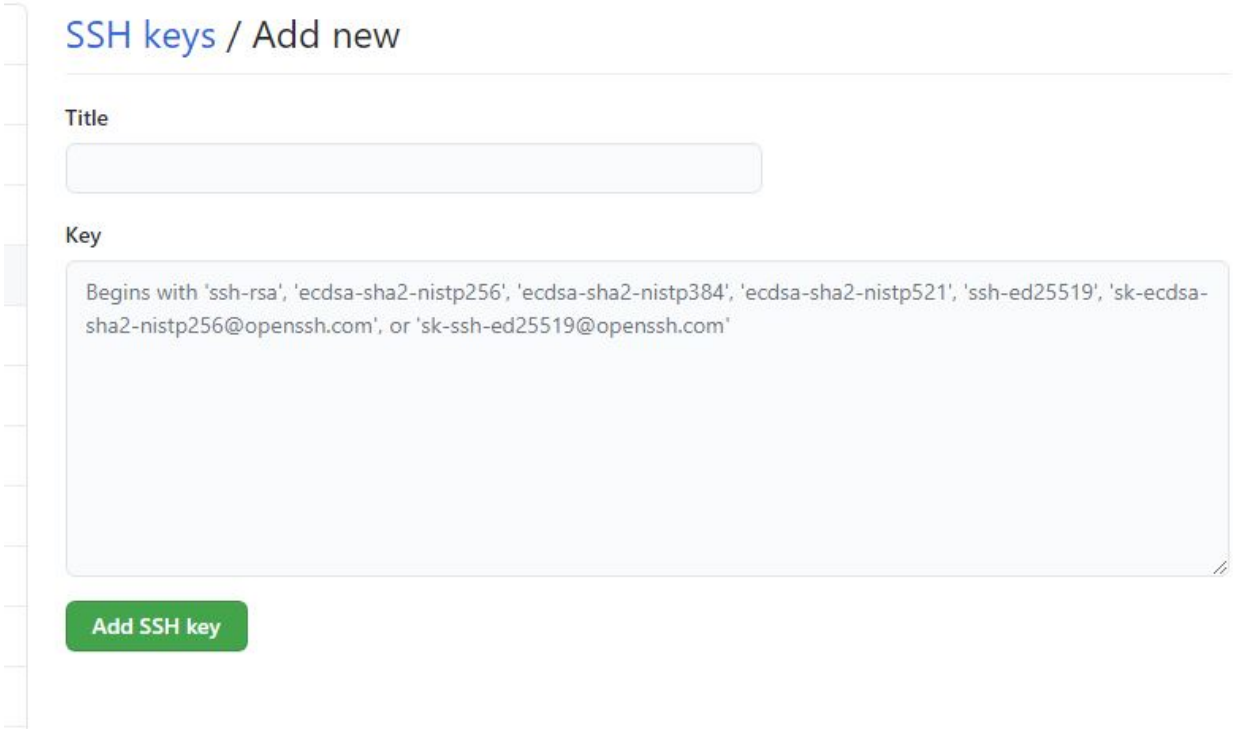
4) **Select SSH and GPG keys Option**



5) After Seleting SSH and GPG keys Option, You can see the following



6) Click on New SSH Key then You can see the following



7) Provide the following

- title myjlc-ssh-key-1
- Key paste the Generated Public SSH key **(Generated in Step1)**

And Click on **Add SSH Key** button

8) Now You can Access Github with SSH from your computer

15. Setup Remote Repository

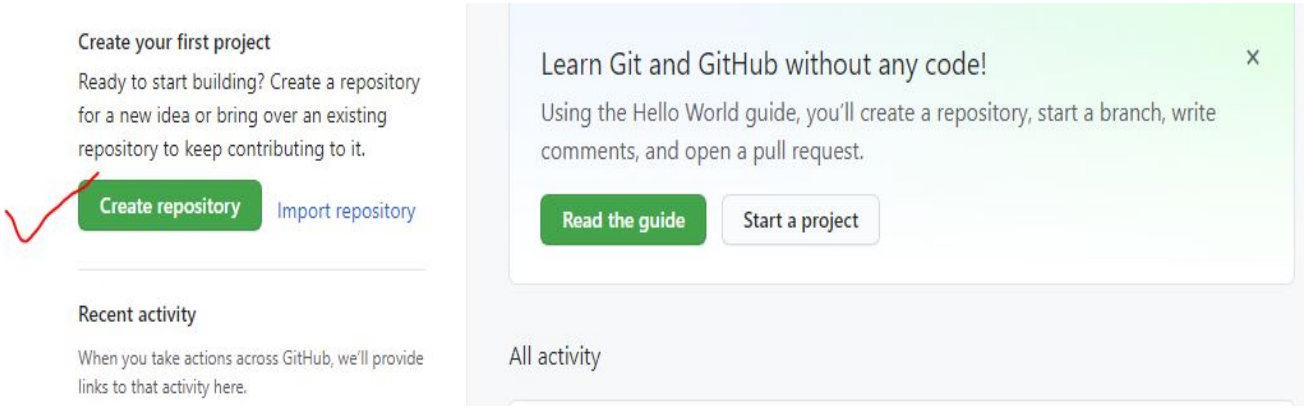
Architect Tasks:

Task 9: Setup Remote repositoty

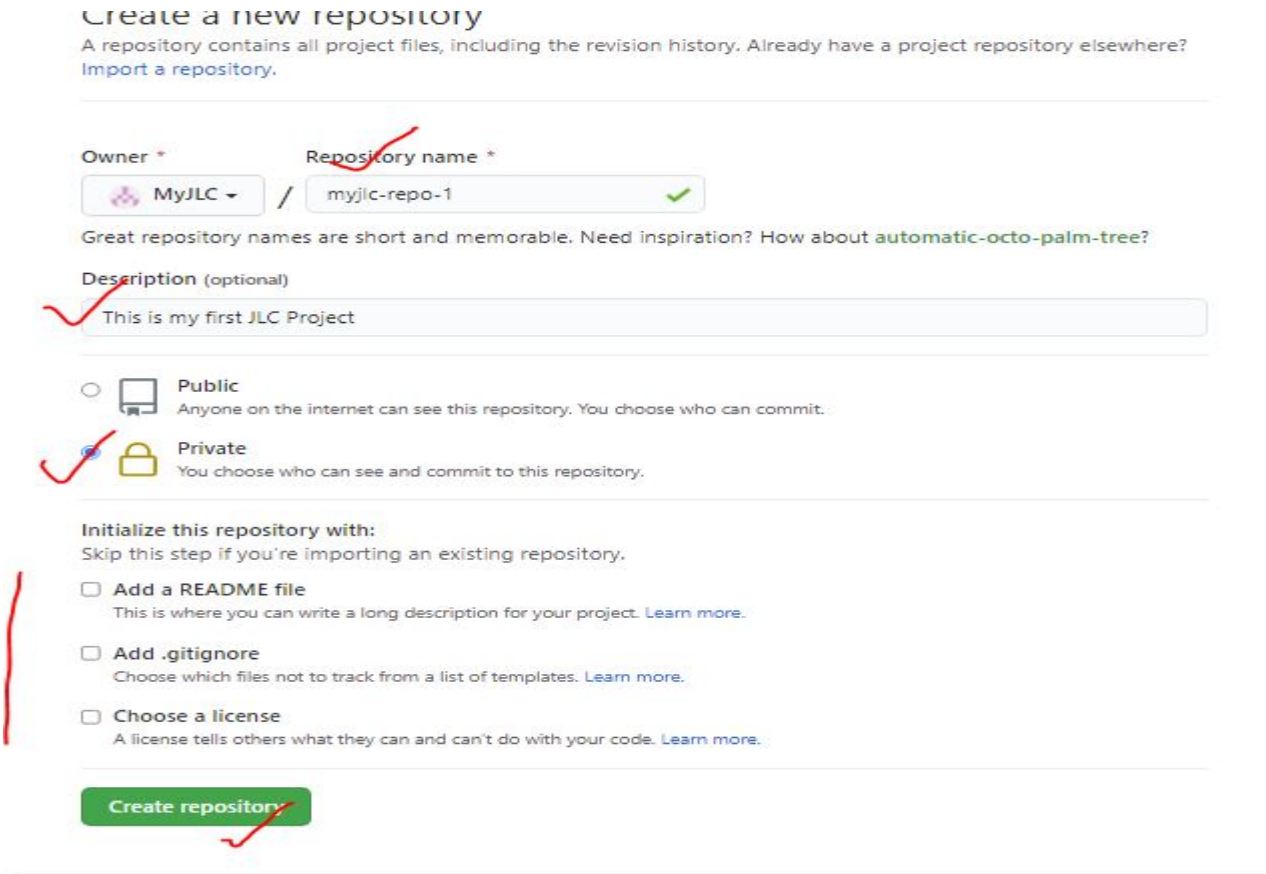
In this Task, You will do

4) Create Empty Remote Repository in GitHub
5) Create the Local Repository
6) Add the Remote Origin to local Repository
7) Push to Remote Repository
8) Doing Commits to local master
9) Pushing to Remote master

1) Makesure that GitHub account is created.
2) Makesure that SSK public Key is added to GitHub Account.
3) Create Your First Empty Repository
   A) Click on Create New Repository



B) Provide the following and Click on Create Repository



4) Do the following in your local Machine

```
mkdir myjlc-repo-1

cd myjlc-repo-1

echo "# myjlc-repo-1" >> README.md

git status

git init

git status

git add README.md

git status

git commit -m "my first commit"

git status
```

5) Change the Branch name to **jlcmaster**

```
git branch -M jlcmaster
```

6) Add the Remote Origin

```
git remote add origin git@github.com:DandesClasses/myjlc-repo-1.git
```

7) Push to remote first time

```
git push -u origin jlcmaster
```

8) Write new File , commit and push

write Hello.java

```
class Hello{
public static void main(String as[]){
System.out.println("Hello Guys !!!");
}
}
```

```
git add Hello.java
git commit -m "my second commit"

git status

git push
```

16. Clone Remote Repository

Developer -1 Tasks:

Task 10: Clone Remote repositoty

In this Task, You will do

1) Clone the Repository
2) Doing Commits to local master
3) Pushing to Remote master

1) Makesure that GitHub account is created.
2) Makesure that SSK public Key is added to GitHub Account.
3) Clone the Remote Repository

```
mkdir developer-1
cd developer-1
git clone git@github.com:DandesClasses/myjlc-repo-1.git
```

4) Write your Code and Commit

```
echo "I am Demo1.java">>Demo1.java
git add Demo1.java
git commit -m "adding Demo1.java by dev-1"
```

5) Push to Remote Origin

```
git push
```

17. Clone Remote Repository

Developer -2 Tasks:

Task 11: Clone Remote repositoty

In this Task, You will do

1) Clone the Repository
2) Doing Commits to local master
3) Pushing to Remote master

1) Clone the Remote Repository

```
mkdir developer-2

cd developer-2

git clone git@github.com:DandesClasses/myjlc-repo-1.git
```

2) Write new File and Commit

```
echo "I am Demo2.java">>Demo2.java

git add Demo2.java

git commit -m "adding Demo2.java by dev-2"
```

3) Update existing File and Commit

```
update Hello.java

  System.out.println("Hello Guys - Update by Dev2");
   System.out.println("Hello Guys - Update by Dev2");


git add Hello.java

git commit -m "Hello.java Update-1 by Dev-2"
```

4) Push to Remote Origin

```
git push
```

17. Pull and Push

Developer -1 Tasks:

Developer-1 has already Cloned the Project and Doing the Commits..

Now Developer has pull the Project before push which may cause the Conflicts if the Same File is Updated by Developer-2

The Developer 1 has to Resolve the Conflicts manually and then push the Code

Task 12: Pull and Push

In this Task, You will do

1) Doing Commits to local master
2) Pull
3) May give conflicts
4) Resolve the Conflicts manually
5) Pushing to Remote master

1) Update Hello.java and Commit

cd developer-1

update Hello.java

```
System.out.println("Hello Guys -  Update by Dev1);
System.out.println("Hello Guys -  Update by Dev1);
```

git add Hello.java

git commit -m "Hello.java Update by Dev-1"

2) Pull the Code from Remote

git pull

3) You can find the Conflicts with Hello.java , it looks like

```
class Hello{
public static void main(String as[]){
System.out.println("Hello Guys !!!");
<<<<<<< HEAD
System.out.println("Hello Guys -  Update - Dev1);
System.out.println("Hello Guys -  Update - Dev1);
=======
System.out.println("Hello Guys -  Update1 - Dev2);
System.out.println("Hello Guys -  Update1 - Dev2);
>>>>>>> f3107de9fe63e0fe1987246dc8c9f77648208a68
}
}
```

4) After Removing the Conflicts with Hello.java, it looks like

```
class Hello{
public static void main(String as[]){
System.out.println("Hello Guys !!!");

System.out.println("Hello Guys -  Update - Dev1);
System.out.println("Hello Guys -  Update - Dev1);

System.out.println("Hello Guys -  Update1 - Dev2);
System.out.println("Hello Guys -  Update1 - Dev2);

}
}
```

5) See the Status , It will be in Untracked State

```
git status
```

6) Add and commit

```
git add Hello.java

git commit -m "Hello.java after Resolving Conflicts"
```

7) Push the Code to Remote

```
git push
```