**Presidential Initiative for Artificial Intelligence & Computing (PIAIC)**

**Lesson-1 : Introduction to Git**

* 1. **:** Introduction

This course will be taught by Mr. Zeeshan Hanif. The book he will use during this course will be *“ Learn Version Control with GIT ”* by Tobais Gunther. This book is a step by step guide for the beginners.

* 1. : Version Control System (Part I)

A version control system is a software utility that tracks and manages changes to a file system. This system is being used to manage the changes in documents, computer programs, large websites and other collection of information over a period of time. So that the changes made to a system, document or program can be tracked.

* 1. : Version Control System (Part II)

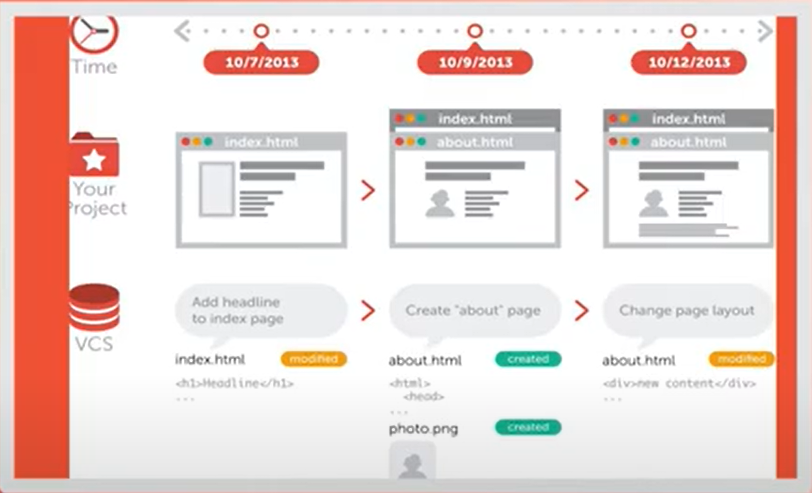
Version Control System is also known as revision control or source control system. It is tightly coupled with software development. All software development is done through Version control system so that it is noted when and what change was incorporated in a system. It is also helpful when we want to discard some changes and move back to a particular state/version.

* 1. : VCS is a king of Database

Version Control System(VCS) can be seen as a database as it stores the snapshot of the document in the software system at that point in time whenever you submit changes in the VCS

* 1. : VCS Timeline

Consider this image of a VCS Timeline. Initially you created a project, added file in it after some point in time you added more files and then made changes in the document. You can store changes to VCS any time, it can be once in a day or multiple times in day or once in a week. It depends on us on what and which change we may want to store the change.



* 1. VCS is Independent

Version control system is independent of the kind of project / technology/ framework you are working with. It works just as well for an HTML website as it does for a design product or an iPhone app. It lets you work with any tool and does not care what kind of text editor , graphics program, file manger to other tool you use.

* 1. Why to use VCS : Collaboration

Without a VCS in place you are probably working together in a shared folder on the same set of files. And you have to coordinate with others so that they don’t work on the same file, it will be very difficult to manage. With a VCS, everything on the team is able to work absolutely free on any file at any time. The VCS will later allow you to manage all the changes into a common version.

For example three of your university fellows are working on a project on three different portions. After few days you need to merge all these three different projects into one project. After few days you again want to merge the changes into a single project and it will become more difficult after few weeks to remember and track changes made to the system. Similarly consider if are working in a big development team and on a big project how difficult it would be to track, manage and merge changes between team members and projects. So VCS system in this scenario is necessary and will help a lot in managing changes.

* 1. Why to use VCS : Storing Versions

Saving a version of your project after making changes is an essential habit. But without a VCS, this becomes tedious and confusing quickly. For example you are working on a project. Now after adding a feature you makes backup of the project . So after some time you will have a lot of project that contains different backups of your project. Consider this what will happen if you are working in a team. VCS helps a lot in this case. So everyday whenever you make changes in the project you make changes and then update these changes in the VCS . Now you will only have one copy whereas all other changes are stored in the VCS. The VCS will help you in restoring to a previous version and also to check what are the difference between two versions.

* 1. Why to use VCS : Restoring Previous Version

VCS gives you the facility to restore your project to an older versions of a file or whole project . For example you want to make changes in a project and after making changes you identified that changes are not working correctly so now you want to revert to the state before these changes. VCS helps you in this regards and you can easily restore back to a specific state.

* 1. Why to use VCS : Understanding what happened

Every time you save a new version of your project, your VCS requires you to provide a short description of what was changed. Additionally if it’s a code / text file you can see what exactly was changed in the files content. So if you are working in a team then VCS will let you see what changes are made by whom and at what time.

* 1. Why to use VCS : Backup

A side effect of using a distributed VCS is that it can act as a backup as every team member has a full blown repository of the project on his disk. If your central server break down and your backup drives fail all you need for recovery is one your team mates local Git repository.

It is important to note how the VCS works. There will a central server where all your changes will be stored and all your information / application code will be placed there and all team mates will talk to that and store all the changes on that central server.

* 1. Different types of VCS

There are different types of Version Control Systems. For Example VSS is Microsoft VCS , SVN is Apache SVN and Git is a Linux contributor product.

VSS is Microsoft Visual Source Safe is a source control program for small development projects / small teams. It is discontinued by the Microsoft. One of the issue in that VCS is that whenever you are doing a change in a file, that file gets locked and now one other than you were allowed to make changes in that file. It is not efficient way in todays era where work at a time in done on multiple files.

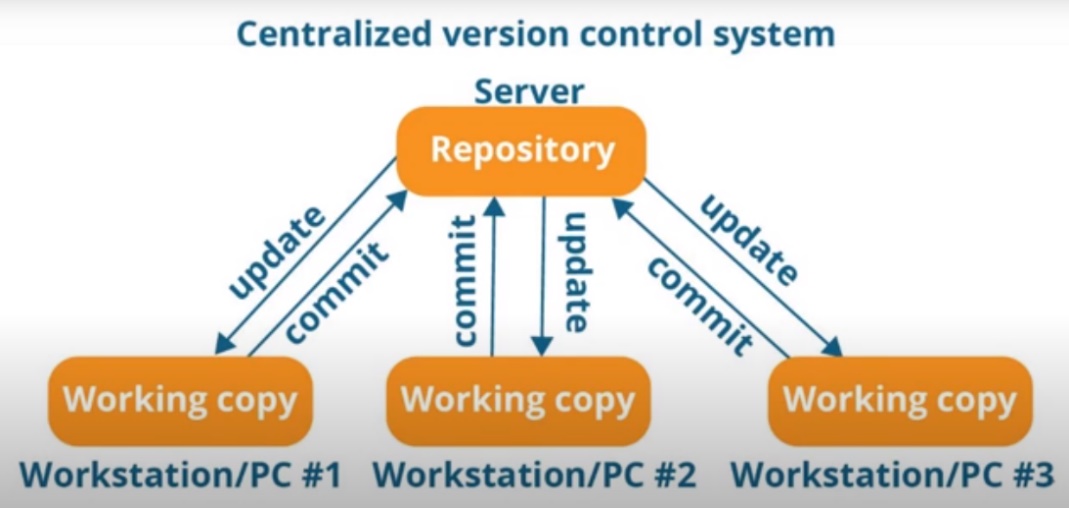
SVN is the abbreviation of Apache Sub Version and is a popular version control system. It is a centralized version control system. It is used for big projects and big teams. But the issue in SVN is that when you work on it every user does not have a complete set of files of the project. All files are present in the central server.

Git is distributed type of VCS for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Every user has a full version of the project code and his changes can be updated on the central server. As every user has full version of the code so in case central server fails any copy of the user can be used as server.

* 1. CVCS (Centralized Version Control System)

There are two types of version control system, Centralized Version Control System (CVCS) and Distributed Version Control System (DVCS). The difference between the two is that in CVCS the main repository is at the Central Control Server and users are connected and users have minimal data required to work.

In case of the DVCS users have full working copy of the code. So all the VCS follow one of these types:



In the above diagram working of CVCS is shown. It is also important to note here that 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 no operation is performed on the local workstation.

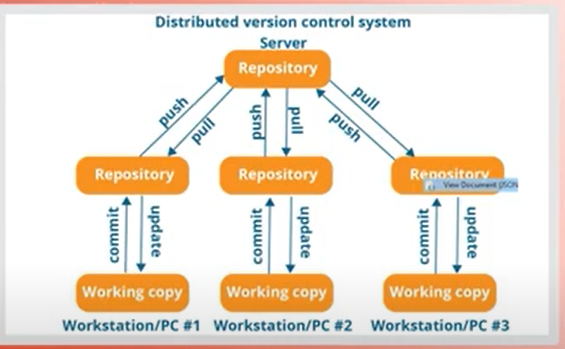
Drawbacks of CVCS

* User does not have a local copy, meaning you always need to be connected to a network to perform any action.
* Since everything is centralized, in case of centralized server getting crashed or corrupted will result in losing the entire data of the project.
  1. DVCS (Distributed Version Control System)

These systems do not necessarily rely on a central server to store all the versions of a project file. Every users has a working copy of the project and users can work on it and can commit changes in the local copy. So snapshots will be created on each commit and user can also track and identify changes. Whenever users wants he can submit the changes to the central server.

In DVCS as every user have complete copy of the project/code so all the files and all the metadata of the files is also available on the user working copy.

Git is a DVCS, so every user has a local copy of the project that keep track of the changes made to the local copy and can update the central repository whenever the user wants .



In the above diagram the working of a DVCS is shown. At the start every user clone the server repository on its system as its working copy. After that whenever user takes changes from the central server it will pull the changes, similarly whenever user send changes on the central server it will be known as push. So in other words we can say that if a Programmer update their local repository with new data from Central Server, then this operation will be known as *Pull* and if the user affect changes to the main repository then that operation will be known as *Push* from the local repository.

For Example : You are working in a DVCS. You make some changes on a file named index.html and commit on your local working copy. Next day you again make some changes and save them on your local repository. Now these changes are being noted / tracked on your local copy of VCS but not on the Central Repository. Now when you push these changes on the central repository than the DVCS will have information about these changes as present on the user local repository.

You need to understand that whenever you are working in a team of 4-5 members, so you need a central server where you can create a repository of project name . Now changes of one member of team which are not pushed to the central server will not be known to other members of the team unless the user push/committed its all changes on the main central repository. So after changes are pushed to a central server these will be available to all other users and will also be transferred to their local working copy whenever they use the Pull operation to get the latest updates/changes.

* 1. Advantages of DVCS (Distributed Version Control System)

1. All operations (except push and 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.
2. 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.
3. 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 effecting changes in the main repository.
4. If central server gets crashed at any point of time, the lost data can be easily recovered from any one of the contributor’s local copy.

**Lesson-2 : Installation and Basic Work Flow**

**2-1:** Installing and Setting up Git

In order to work with Git you need to download git from their website ( <https://git-scm.com/downloads> ). So open the mentioned URL and download the git version according to your OS.

It is important to mention here that Git works on command prompt, so in order to use a GUI tool you need to install the SmartGit GUI tool. You can download the SmartGit GUI tool from the website ( <https://www.syntevo.com/smartgit/download/> )

Now first of all install Git with default settings. After the installation go the command prompt and write git and press enter. If some information related to git is visible then it means Git is installed on your system.

After the installation of the git you need to set the name and email address , so that Git knows who is the author of this particular snapshot or person making changes. You can set them on command prompt using commands :

* git config --global user.name “Name”
* git config --global user.email “You Email Address”

**2-2:** Installing and Setting up SmartGit

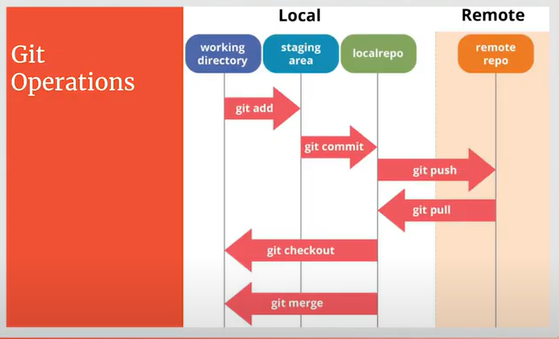
After installing Git we need to install the SmartGit so that we can have a GUI interface to manage the git repositories. Install the SmartGit with default options. After the installation SmartGit will run and ask for following options :

1. On the 1st screen system will ask for the License Information. Check the I understand and agree all terms and conditions checkbox and press Next.
2. On the 2nd Screen system we need to specify the path of the Git Executable. By default SmartGit try to find itself and specify the path
3. On the 3rd Screen it will ask for User Information which it will get automatically
4. On the 4th Screen it will ask for SSH Client and then select for the default options
5. On the 5th Screen system will ask for the Style, You can choose the style you want we can select Standard for the time being
6. On the last screen software will ask for the privacy setting we will leave it as default and press finish
7. Then system will ask to evaluate the lenience . After online evaluation copy the licience code on the given text box and start the SmartGit

If there is already some git projects in your folder then system will show them otherwise it will show blank. From this we will start using git operations.

**2-3:** Git Operations

Let us see some basic Git Operations . There are 5 basic git operations : Initialize , Add , Command, Pull and Push. There are also some advanced operations like branching , merging and Rebasing.



In the above diagram visualizations of the basic operations is shown. First of all there is a working directory where you place your files. So first of all you will do the initialization which is done using git init. Now suppose you have added some files in it and changed some existing files data. Now after that you can commit all or can select the files that you want to commit on the server . Git provides the option to either commit all changes on the local repository or commit the selected files using git add.

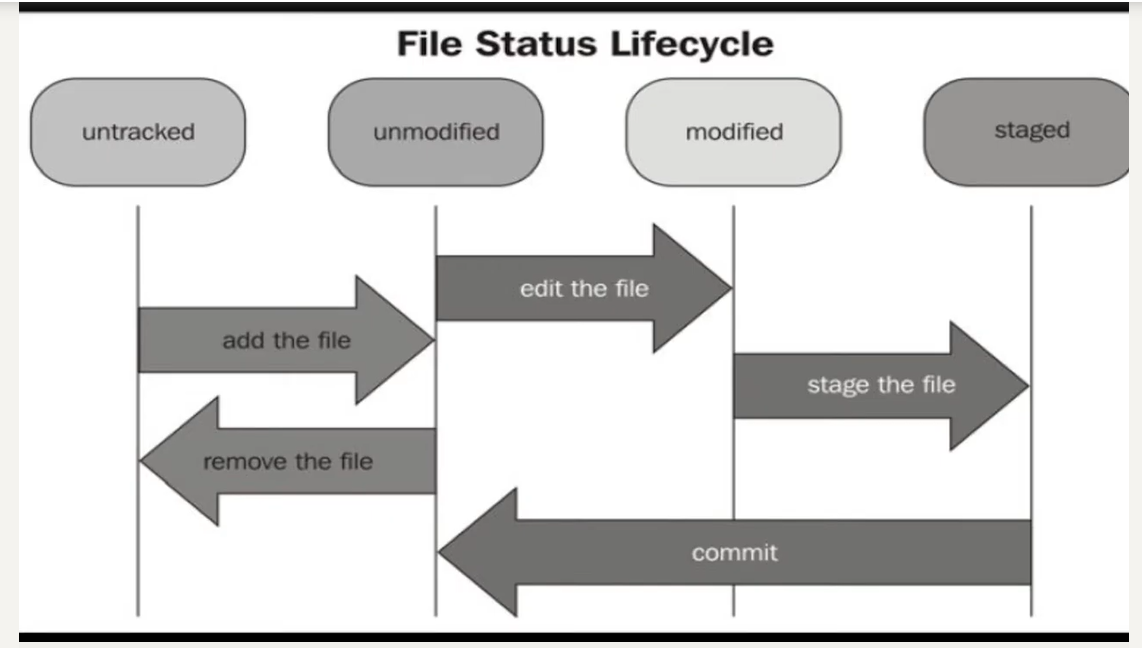
Git add sends the selected files to staging area and then when user commit the changes then only files at the staging area will commit on the local repository and it will create a snapshot of changes. From local repository it is up to user when he wants to push these changes on the remote repository or to Pull the changes from the Remote repository.

Similarly on your local repository if you want to go to different branch you can use the git checkout or if you want to merge then use the git merge options .

**2-4:** Important Terms

Let us discuss some important terms that will be frequently used in the Git.

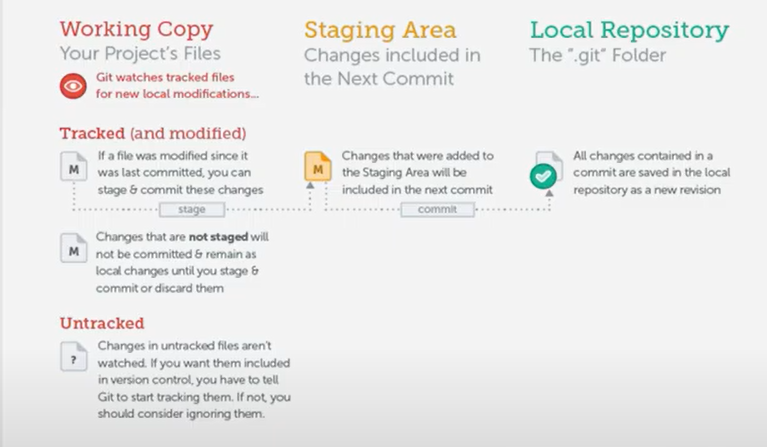
1. Repository: It is a kind of database where your VCS stores all the versions and metadata that accumulate in the course of your project. Whenever you directory becomes a git repository then a hidden folder “.git” will be created there that will contain all the history, commits and versioning all included in it. The “.git” also indicates that it is a git repository
2. Working Directory: The root folder of your project is often called the Working Copy” or “Working Directory”. It is the directory on your local computer that contains your project files.
3. File Status : The files inside your working directories have some statuses which are as under:
   1. Untracked
   2. Unmodified
   3. Modified
   4. staged



File at the start is untracked. Whenever you create a file in the working folder it is untracked file. So git will not track changes for that file. So if you want to track that file then you will use git add option to add the file in the git tracking system. Now the file status will be changed from untracked to unmodified. After that if you edit the file its status will become modified. Now you can use git add command to place this file in the staging area and then commit the changes in that file. After the changes are commit the file status will again become unmodified as all changes are committed in the repository and now file status is unmodified. The file status will remain unmodified until you make changes in the file.

1. Staging Area : It is a virtual place that collects all the files you want to include in the next commit. In git simply making some changes does not mean that you are automatically committed. So it is necessary to move changes to the staging area first then these files will be committed in the local repository.

Please consider the image below. In Working Copy there are two types of files : untracked and tracked. Whenever you create a file in the git working copy by default this file is untracked as this file is not part of tracking so git will not track changes in this file. So whenever you add file then this file status becomes tracked and now git will track all changes made to this file and after that you can move them to staging area and then commit them to make them part of the repository.



**2-5:** Basic workflow

Let us discuss the basic workflow of the Git. We will cover different scenarios .

1. Starting with an Unversioned, Local Project

Open terminal and create a directory on your machine. For Example:

C:\Repo\myproject

1. Open terminal window / command prompt and go to that folder
2. Initialize repository in this directory
   1. git init : This command will initialize the git repository in this folder so git will start tracking this folder. This will also create .git hidden folder in your myproject directory which will make your current folder a git repository
3. Now create files in that directory e.g: first.txt and second.txt
4. Check Status , it will show you two untracked files . You can check the file status using git status
5. Add these files to the staging area. There are two ways to add files in staging area:
   1. ***git add first.txt second.txt***  (This will add these two files in the staging area)
   2. ***git add .*** ( This will add all the files in the current folder in the staging area and mark them for tracking)
6. Now to commit these files in the git to register them for snapshot as:
   1. ***git commit -m “Implemented new feature”***
7. Commit message is very important. You should provide proper commit message so that it can be refer back to identify what was added in that commit.
8. You can check logs to see the commit history using command
   1. ***git log***

So now you can add / track / commit files as mentioned above.

**2-6:** Basic workflow : Demo with Terminal

Let us do some practical work . we will start with a basic example.

First of all create a folder on your hard drive. We can name it MyRepository. In this folder I will place all my projects. Now create another folder for the first project as firstproject. Open terminal window in that folder by writing cmd in the top bar of File Explorer while this folder is selected in File Explorer. It will open terminal window / command prompt in that folder.

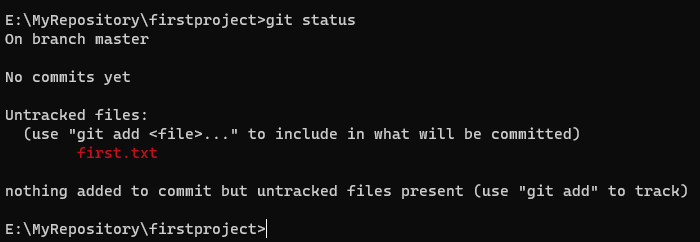
Step 1: The newly created directory is empty and it does not have any connection with git. So first task is to register this folder with git using command ***git init***

When this command is successfully executed it will return a message “ *Initialized empty Git repository in E:/MyRepository/firstproject/.git/* “

Now this means git repository is created in this folder. A hidden folder***.git*** is created in that folder whenever the git is initialized in a folder and it is an indication that this folder is a git repository. The ***.git*** folder stores all the information regarding version , history of changes etc.

Step 2: We can check the status of git using the command ***git status .*** This will show the status of this git Repository. As the folder is empty it will show following information:  
*On branch master  
No commits yet  
Nothing to commit (create/copy files and use "git add" to track)*

Step 3: Let us create a file in the folder named first.txt. Write something in it and save it. Now if we check the status of the file using ***git status*** command it will show us that there are some untracked files in it.

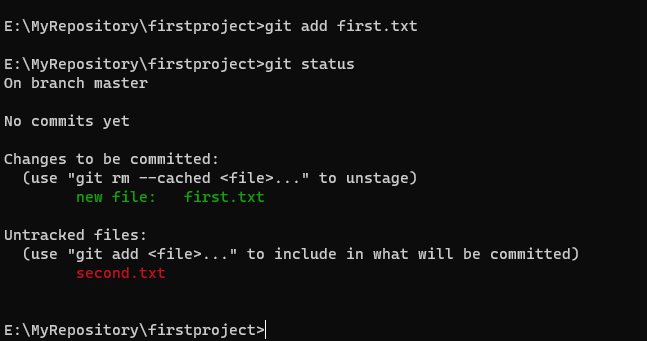
**

Now git is telling us that there is a file named first.txt which is not tracked and git cannot store/track changes in that file. Similarly if we add another file second.txt in it and again get the status git will show us that there two untracked file.

Step 4: We can add file(s) in the git in two ways:

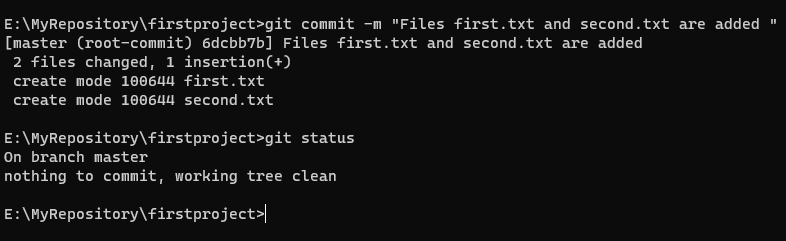
1. ***git add . (***This will add all untracked files in that folder in the git repository for tracking***)***
2. ***git add first.txt second.txt*** *(This will add these two files in the git repository)*
3. ***git add \*.txt*** *(To add all untracked files have txt extension to add in repository for tracking)*

Now if we add using command ***git add first.txt*** , git will add only first.txt for tracking.

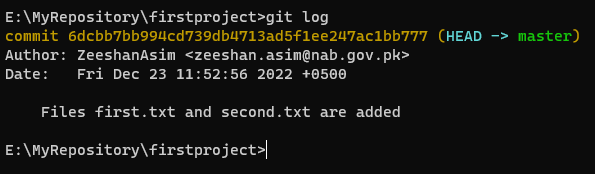


From the above figure it is visible that files being tracked are shown in green where are untracked files are in reddish color. Similarly also add the second.txt file for tracking. Now if we get status there will be no untracked files.

Step 5: We can commit the changes in the git using the command ***git commit -m “Files first.txt and second.txt are added “*** . Please note that in this command we have to specify a message. This message should be descriptive one so that people can know the reason for this commit.

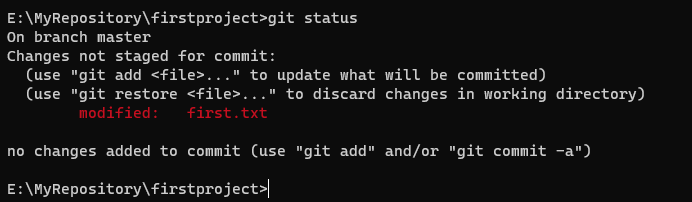


Step 6: We can also check the log using command ***git log***



Log showing us that we are in the master repository. Author is Zeeshan Asim and commit time along with the commit message is shown

Step 7: Lets make some changes in a file first.txt. After making changes save the changes. Now if we view the status of file it will show us that a file first.txt is modified but this file is not staged for commit. It is due to the fact that after commit we have not added the file first.txt for stage after commit



Lets also make changes in the second.txt. Now if we want to add only first.txt file for staging we will use git command ***git add first.txt***

Now if we commit at this time as ***git commit -m “update first.txt”***  it will update all the changes in the repository and create a snapshot of this. Now at this stage although changes were made to both the files but changes only to first.txt are committed as we only added first.txt file on the staging area using git add command

**2-7:** Basic workflow : Demo with UI

Let us work with GUI tool for Git. First of all create / select the folder where you want your repo should exists. Now start the SmartGit GUI tool, from Repository menu select Add/Create option. Now select the folder to be used as Git Repository. Now as soon as you select the folder SmartGit checks either this folders is already configured as Git Repository or not. If it is not configured SmartGit will show the message to initialize it as Git Repository. You can also check this folder using File Explorer and there will be a hidden .get folder in it which means this folder is a Git Repository.

Now if we create some files using file explorer and then open the SmartGit you will see that file there and its status will be untracked. In this status file will not be tracked for changes. To track the file we will mark this file for stage. This is same as we do using git add command at terminal window. Now this files in being tracked . If you now make changes in the file these changes will be tracked .

Now after making changes if you are satisfied with the changes you can commit it in the database using commit option. You need to specify the commit message and then press the commit button to commit the changes. One more benefit of Smart git is that you can directly commit an untracked file . The SmartGit will not only add that file in the repository but will also commit it in the repository. But if you are working on the command prompt / terminal window you have to do the git add and commit commands .

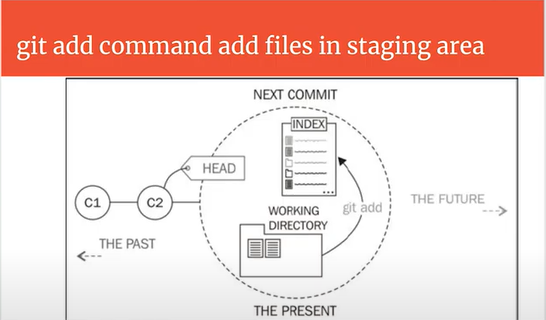
So if we add more files in the repository and make some changes in some files then we can make changes in these files and commit changes in the repository. The changes can be viewed in the SmartGit and the editor will highlight the changes.

**2-8:** Basic workflow : Demo Terminal UI Mix

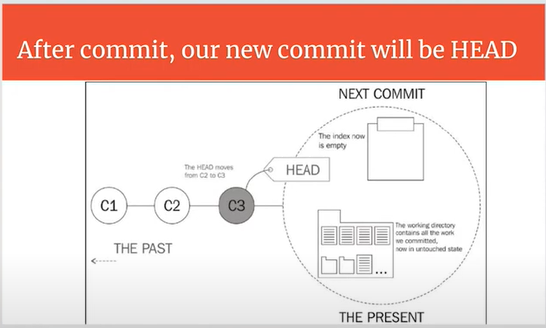
In the last example we have seen working using SmartGit GUI editor. All the changes we have made in the GUI can be viewed from the terminal / command prompt as the SmartGit GUI tool is built over the Git working in command prompt. For example : using git status command you can view the status of files. Git log can be used to view the log of changes.

**2-9:** Commit : Past, Present and Future

In the last lecture we have studied a term head. Head represents the current working version of the Git . Consider the diagram where c1 and c2 are commits and the last commit c2 represents the Head. So the new changes made in files is the part of the next commit.



Now if we commit the new changes (C3) , then Head will represent the C3 as shown in the figure below.



**2-10:** Commit Hash

Git Hash is 40 character unique identifier known as Commit Hash. As multiple people working in parallel, committing their work offline, without being connected to a shared repository. Therefore, commit hash helps in identifying which user made specific commit.

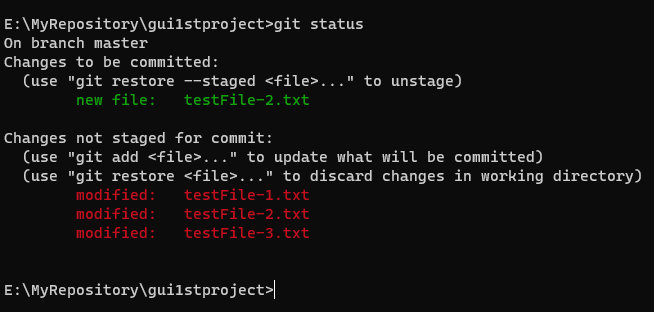
**2-11:** Unstage Files

Git reset command is used to remove files from staging area. Git reset can remove changes in files if they are not committed. This option is used when you mistakenly add a file to staging area. So when we use the git reset all files will be removed from the staging area. So you can again add only those files to staging which you want.

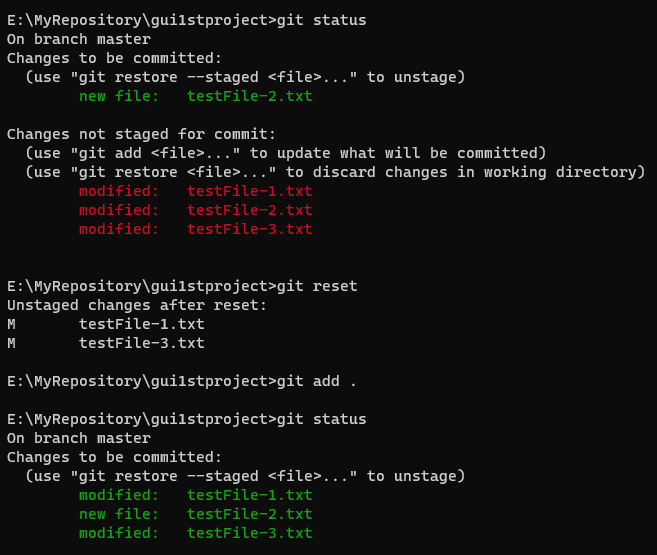
Note if you use command ***git reset*** it will only remove all files from the staging area, but the changes made on these files will remain there. If we use the command **git reset --hard** then the changes we have made in the files present at staging area will also be removed.

**2-12:** Unstage Files : Demo with Terminal

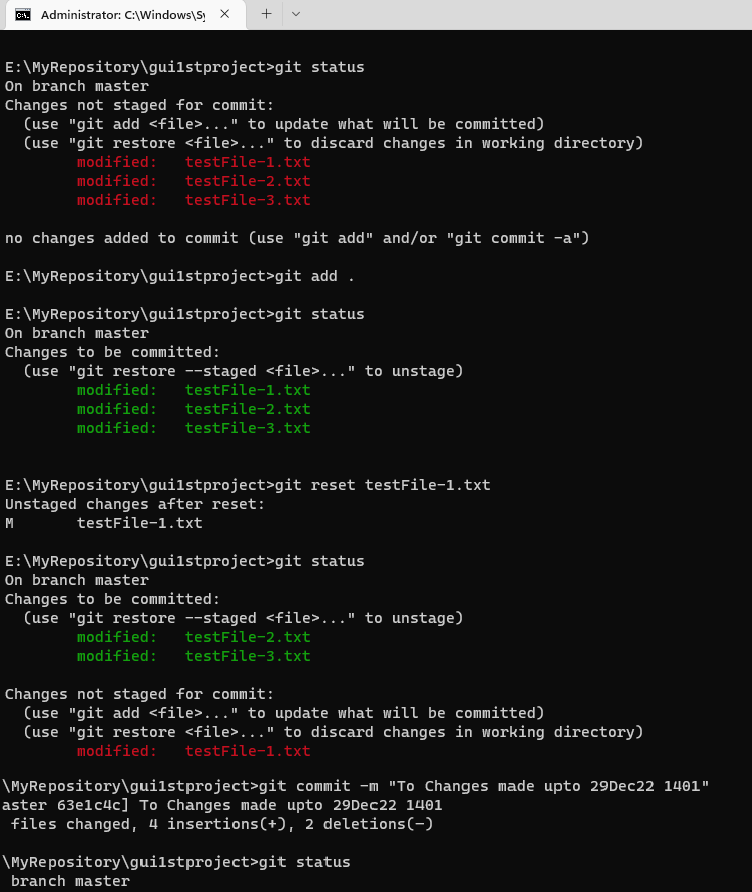
At terminal we can use ***git status*** command to get the status of files in the repository. For example see the picture below to get information about file statuses.



Now if we execute git reset command the command git reset the statuses of files will be:



Now consider you have thee files that are untracked and you mistakenly give the command ***git add .*** that will put all files on the staging as mentioned in the example given below:



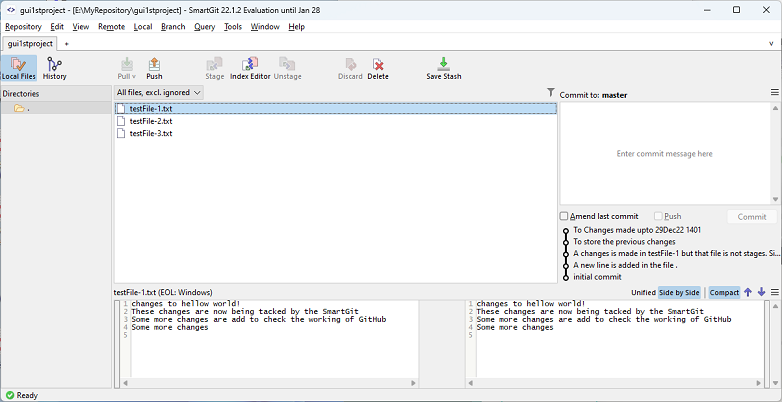
In that case if you want to unstage a file you can use command ***git reset filename*** to unstage a specific file. Similarly you can also use ***git restore --staged testFile-1.txt*** to unstage a file. If you use command ***git reset*** it will reset all files and all files present in staging will be cleared and all files will be marked as untracked.

**Note:** git reset command removes the files from staging but did not reverse the changes made to these file. In case you also want to clear all changes made in files and remove them from stage you can use command ***git reset--hard***

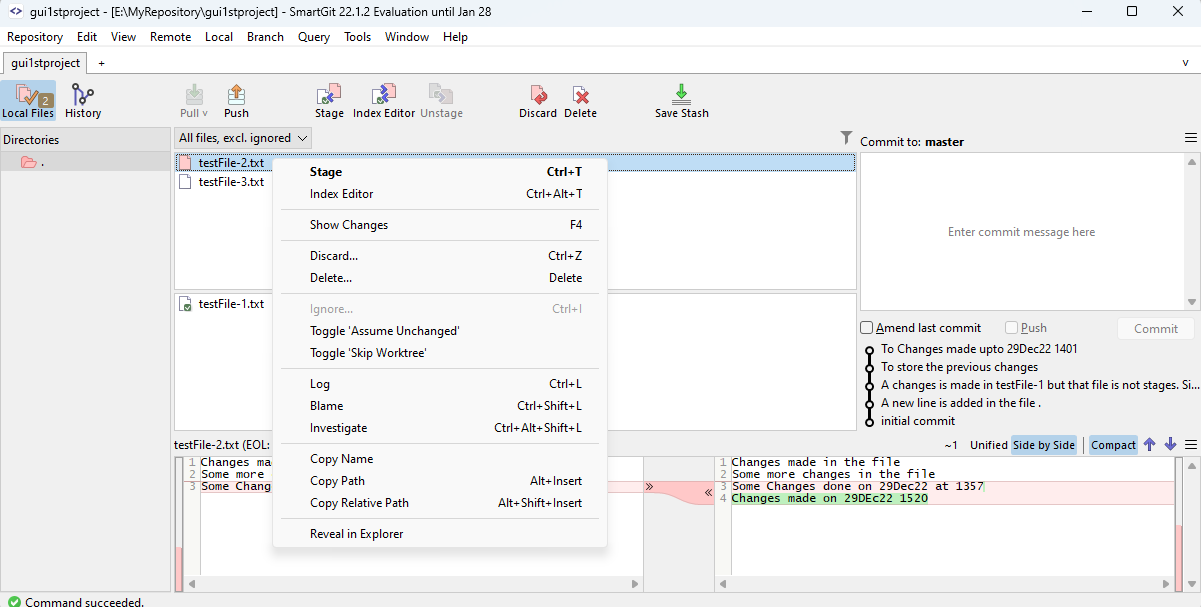
Now to commit the changes in files present at stage we use command ***git commit -m “Commit Message”***

**2-13:** Unstage Files : Demo with UI

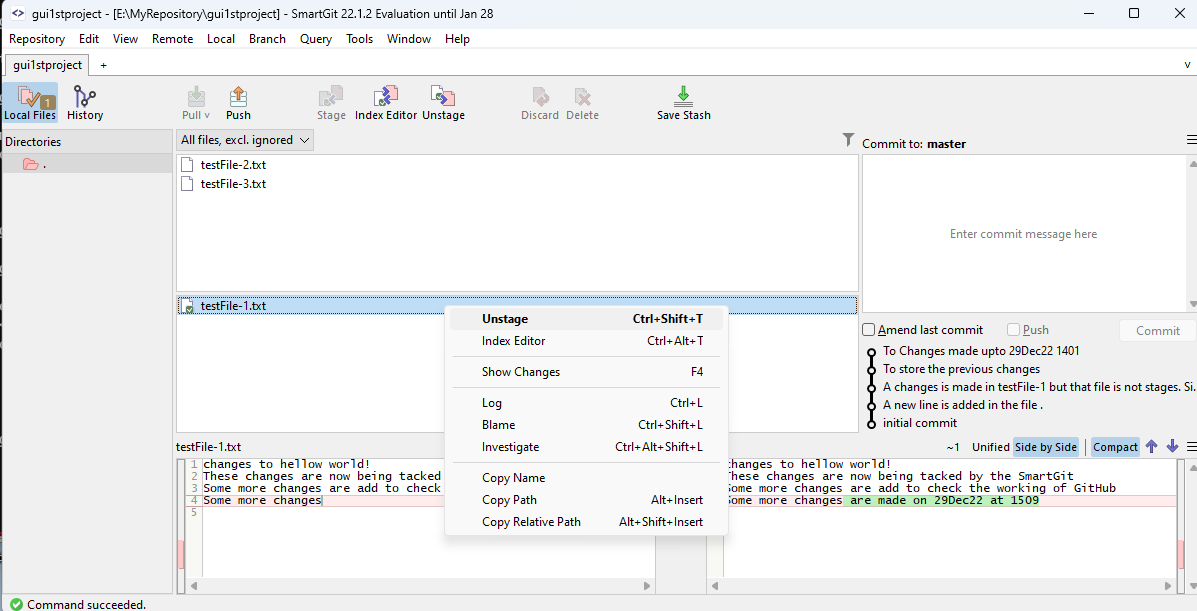
Let us see how can Stage/UnStage files using SmartGit UI tool. Let us consider our current status is:



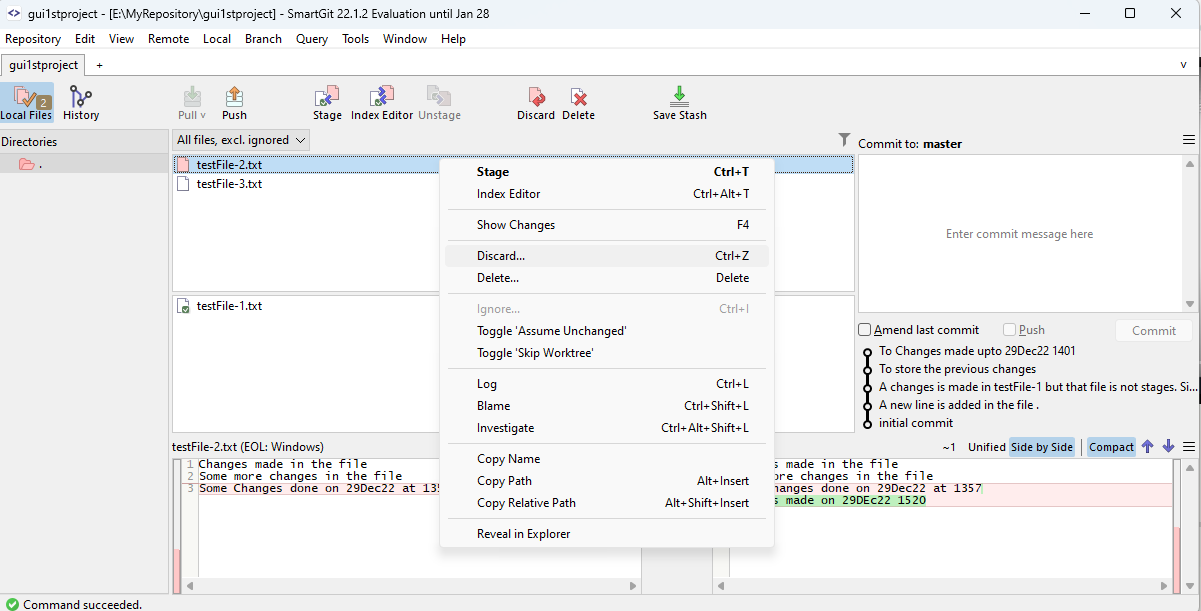
There are three files in the repository and all are untracked. Now make some change in the testFile-1.txt . Now this file contains some changes, in order to track changes for this file we need to stage



that file using SmartGit. Using SmartGit you can stage any file that is changed. You have to select the file and then click on the Stage icon. This will stage this file and now changes made to this can be tracked and committed as shown in the figure below.



You can stage any staged file by selecting the file and by clicking the Unstage option as shown in the above figure. You can reset/remove all the changes made to a file till last command using GUI with discard option as shown in the picture below. This options works same as ***git reset –hard*** to discard all changes till the last commit and unstage the file.



**2-14:** Ignoring Files

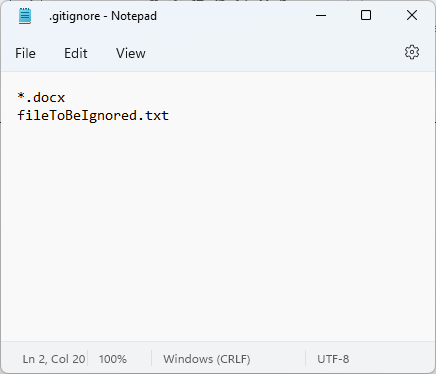
While working on projects there are some files that you did not want to be version controlled. These files are normally User specific or OS specific and you do not want to part of Git History. For example Node\_Modules while working with Node.js, Builds , logs. DS\_Store is a file that is present in MacOS . Means these files will not be tracked for changes.

In order to do this you need to create a file **.gitignore** that contains the rules to ignore files. Add file or directory path or extension or name. For example:

* if you want to ignore one specific file you can mention **path/filename.extension**
* If you want to ignore all files with a certain name (any where in the project) then you can **filename.extension**
* If you want to ignore all files of a certain type (anywhere in the project you can write **\*.extension**
* If you want to ignore certain files in a folder you can write **path/foldername**

**2-15:** Ignoring Files : Demo with Terminal

Let us see how we can ignore some files using terminal/command window. For those first we have to create a **.ignore** file. You have to make sure that there is no extension of this file and file name should be exactly the same as mentioned . Now open this file using notepad or any text editor. Write the file name or extensions or file with complete path as discussed in previous lecture. For Example:



Now if you view the git status for this folder using ***git status*** command. You will see that two files which we mentioned in the **.gitignore** file are shown in the git but the **.gitginore** itself is shown as untracked file. You need to commit this file so that changes made to this files are tracked and it also remain the part of the repository so that in future if someone else pulls this repository same files remain ignored there.   
Note: You can also ignore all files in a folder if you mention the folder name in the **.gitginore**

