**What is Version Control?**

Version control is a system that record changes to a file or set of files overtime so that we can recall specific version later

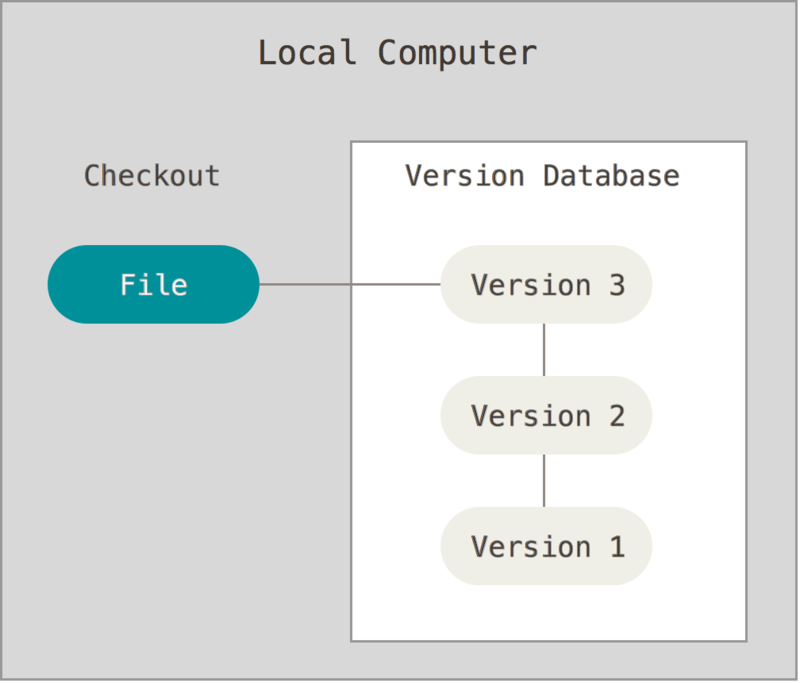
**Local Version Control Systems**

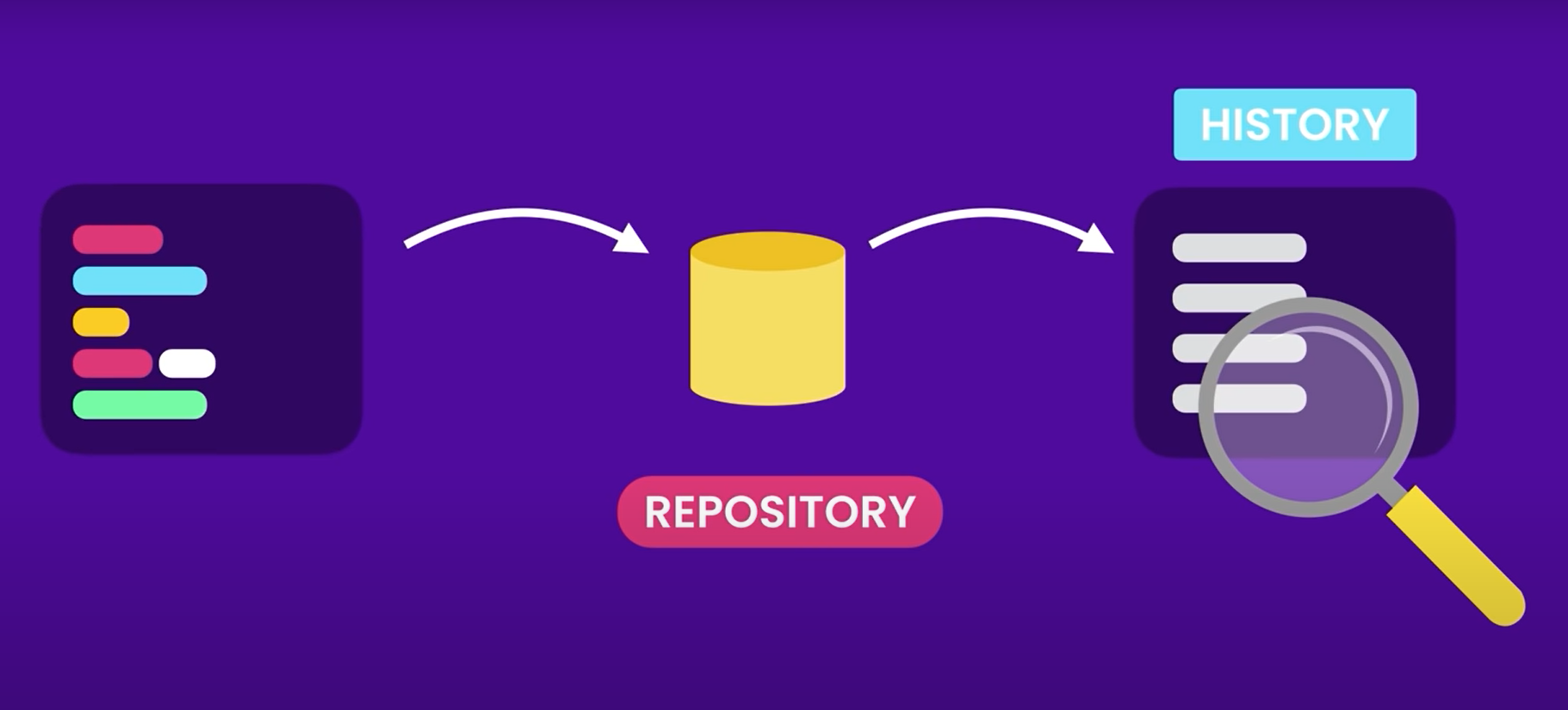
* Copy files into timestamped directory
* Simple
* Error Prone
* Accidentally write the wrong file or copy over files we don’t mean to



**VCS (Version Control System)**

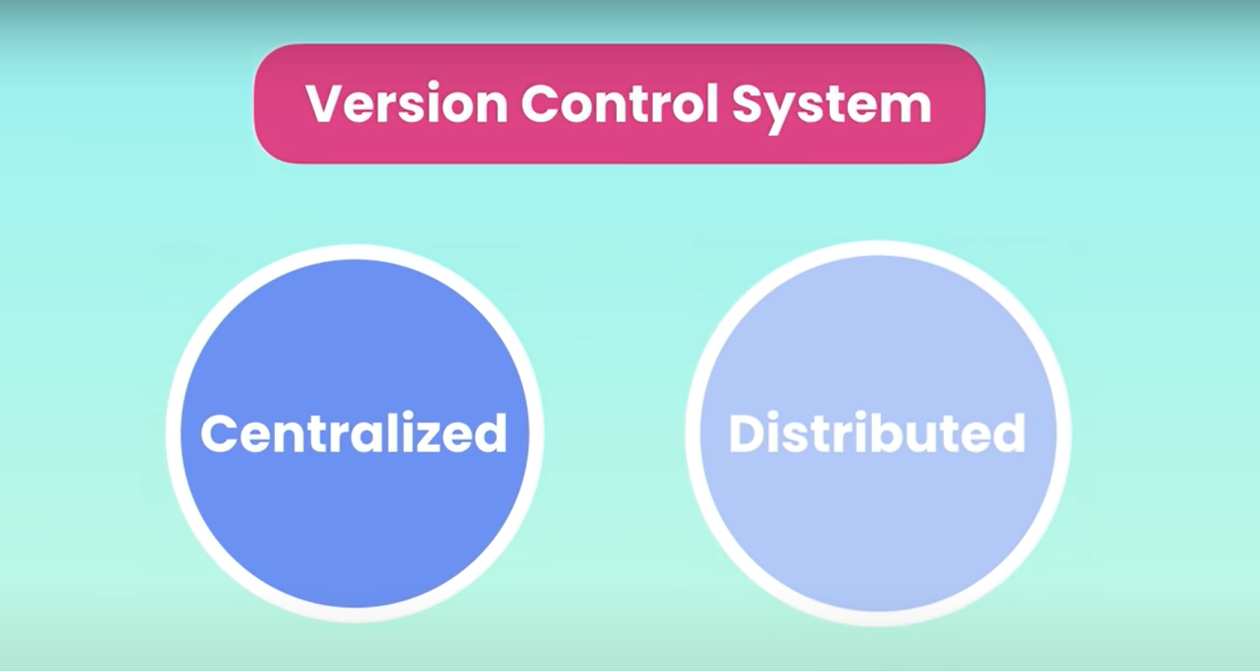
Simple database to keep all the changes to files





**Issue:** Collaborate with developers on other systems

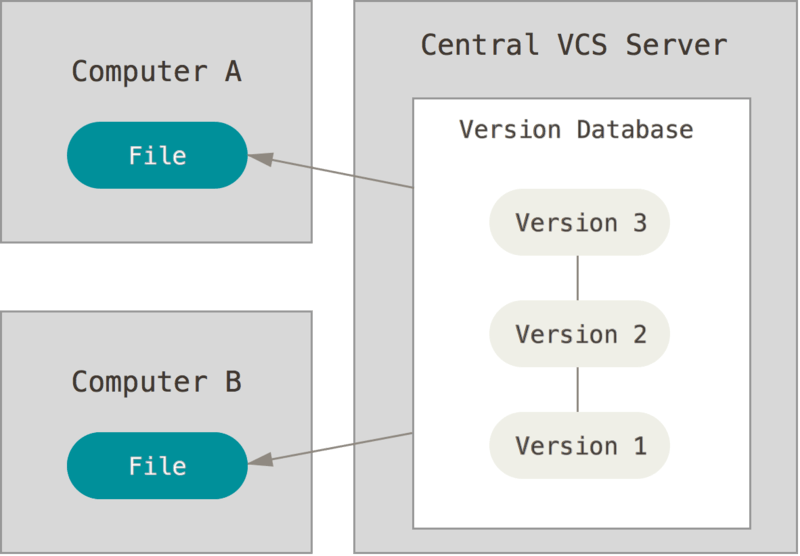
**2 Types of Version Control System**

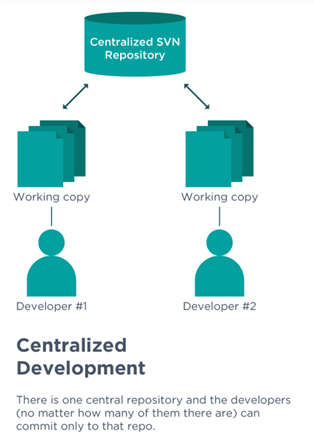


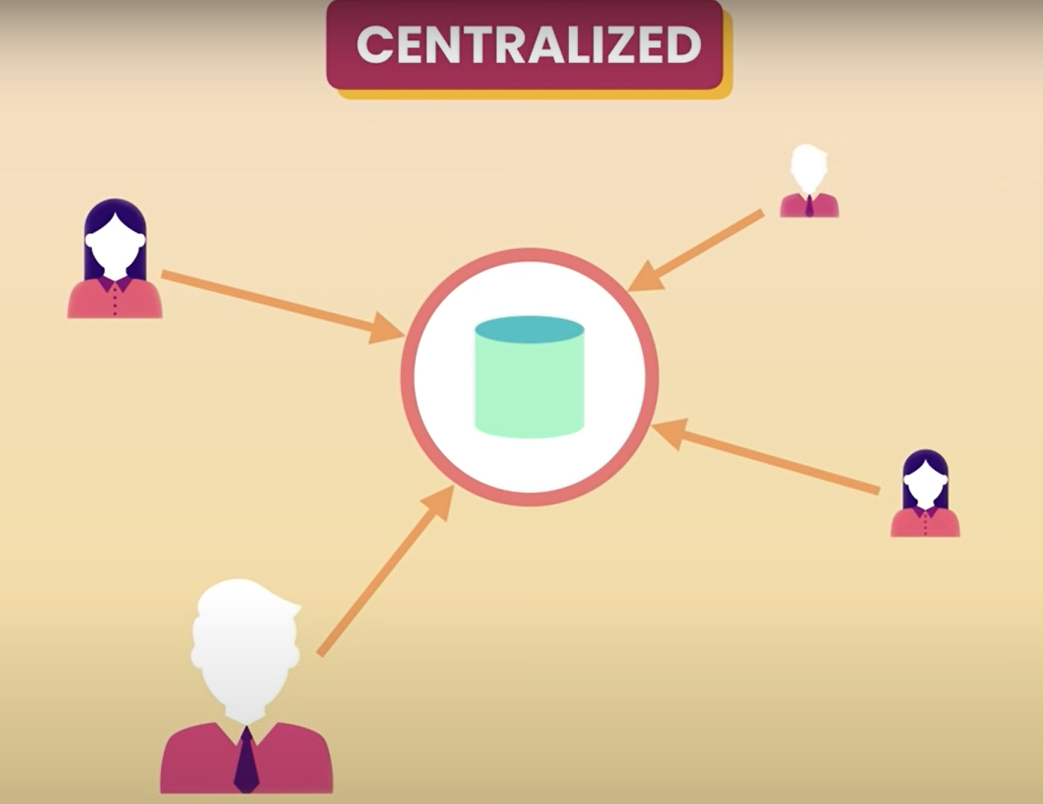
**Centralized Version Control Systems**

To overcome this issue, Centralized Version Control System (CVCS) were developed

Systems like CVS, Subversion and Perforce have a single server that contains all the versioned files







**Advantages on Centralized Version Control System**

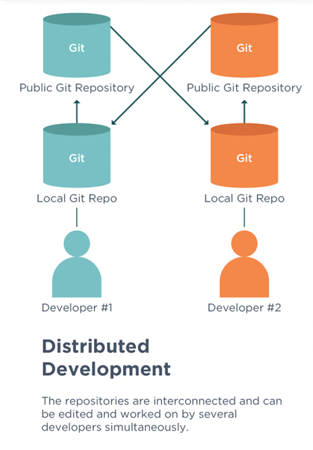
* Everyone knows to a certain degree what everyone else on the project is doing
* Administrator can control who can do what
* All team members connect to the CVS and gets the latest copy of the code
* Share their changes with others
* Ex : Subversion, Team Foundation Servers

**Downside/Pitfall on CVCS**

* Single point of failure

**Distributed Version Control Systems (DVCS)**

****



**Pros:**

* Every team member has a copy of the project and history on their machine
* Save snapshot of the project, locally
* Synchronize our work with others if the server is offline
* Ex: Git, Mercurial

**Why Git**

* Its Free
* Open Source
* Super-Fast
* Scalable
* Branching/Merging is easy and fast
* More than 90% of the Software Projects in the world uses Git
* GitHub, GitLab, Beanstalk, Apache Subversion, AWS CodeCommit, Microsoft Team Foundation Server, Bitbucket, CVS (Concurrent Versions System), Mercurial, Bazaar, Darcs,

**Using Git**

* Command Line
* IDE’s & Code Editors provides built-in support for Git
* Graphical User Interface (Git Bash, GitKraken, SourceTree)
* Supports Windows, Mac, Linux, Android, IOS

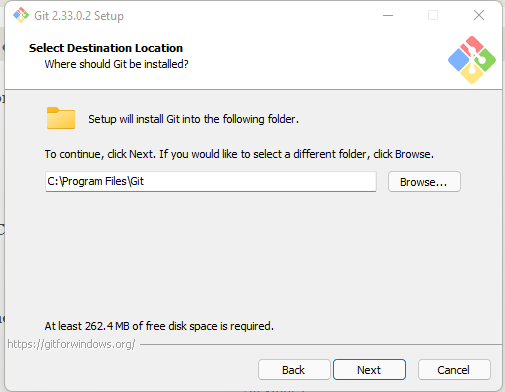
**Why Command Line**

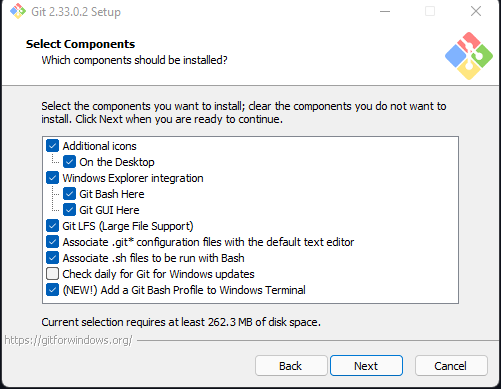
* GUI tools have limitations
* GUI tools are not always available
* Command Line is faster and easier

**Installing Git**

Git Download Link : [Git - Downloads (git-scm.com)](https://git-scm.com/downloads)

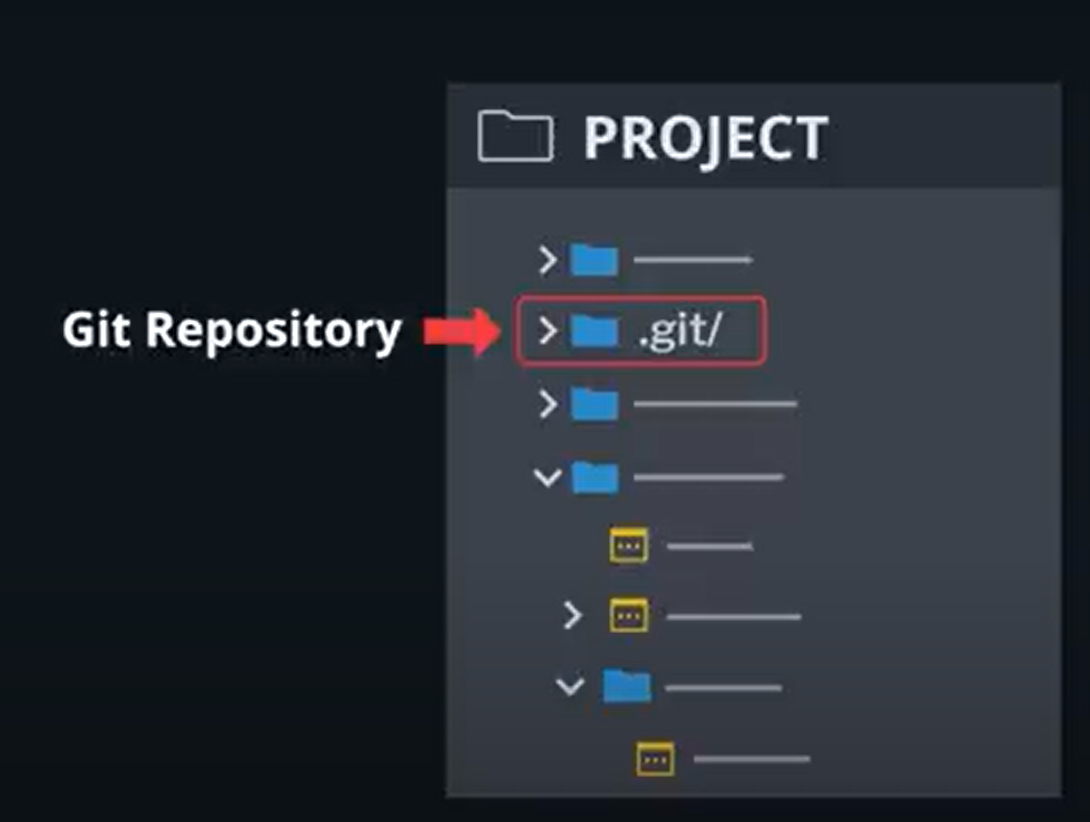
Git GUI Tools: [Git - GUI Clients (git-scm.com)](https://git-scm.com/download/gui/windows)



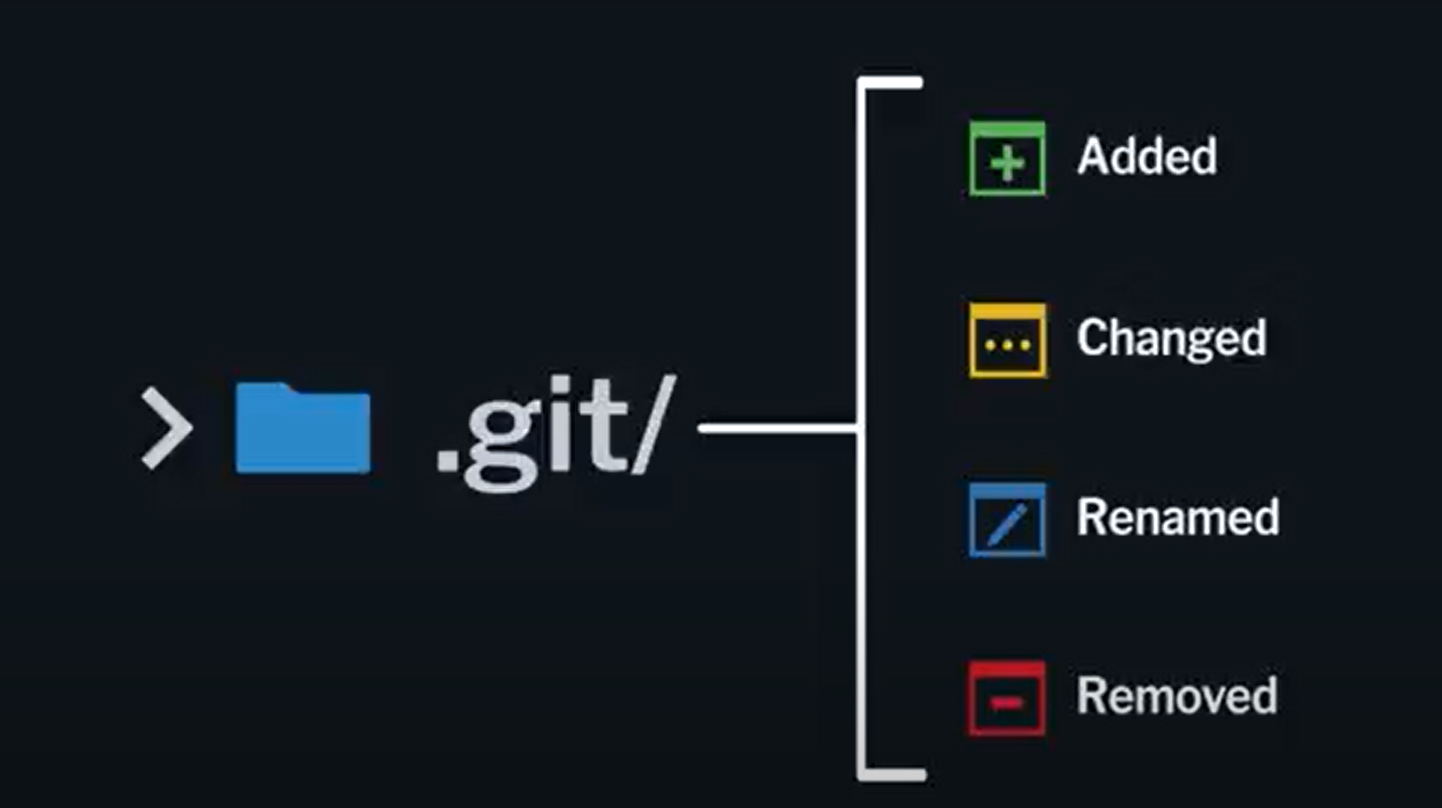


**What is a Git Repository?**

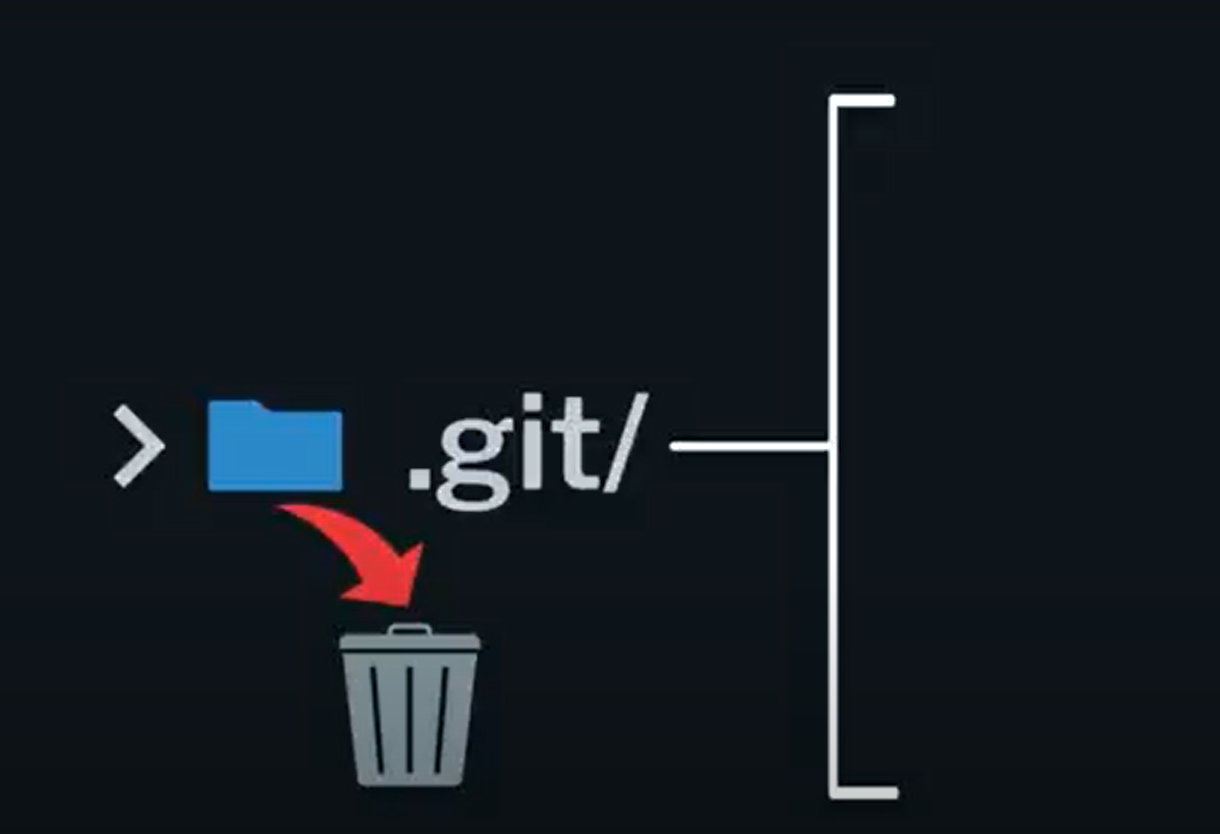
* Git is a program that tracks changes made to the files
* Git once installed is initialized on a project to create a Git Repository
* Git Repository is a .git folder inside the project

****

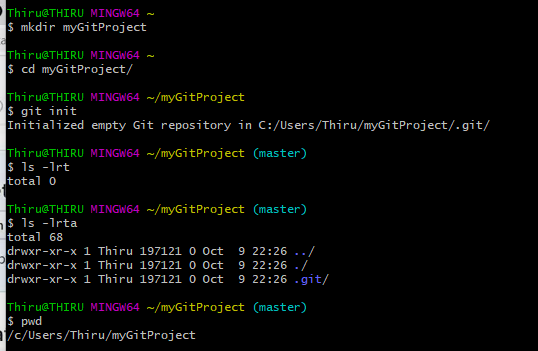
* .git/ repository tracks all changes made to the file in the project
* Builds a history overtime



**Deleting the. git repository, deletes all the files**



**Initializing Git for the first Time**

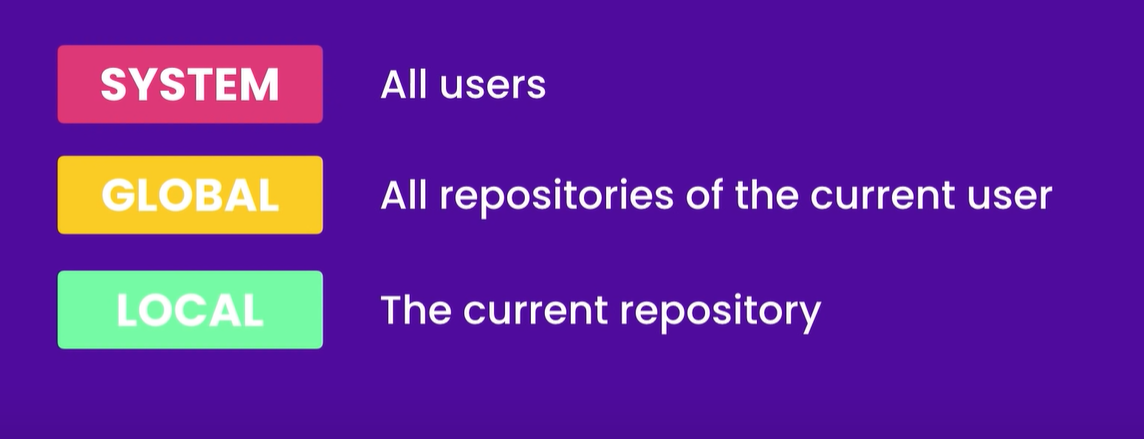
****

**Settings**

* Name
  + git config –global user.name “Thirumal Murugan”
* Email
  + git config –global user.email Thirumal.murugan@gmail.com
* Default Editor
  + git config –global core.editor “vim”
* Line Editing
  + git config --global core.autocrlf true #Windows => true #MacOS => false

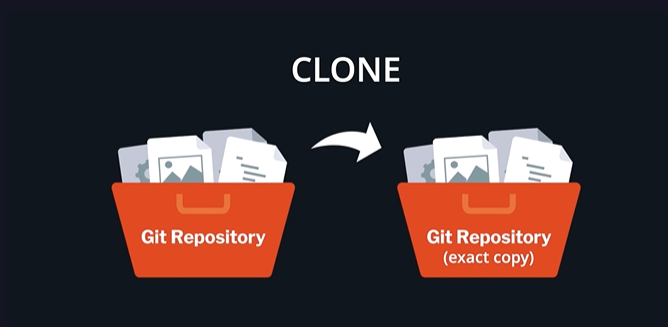
The git config core.autocrlf command is used to change how Git handles line endings

\r\n -Windows \n MacOS

3

**Git Clone**

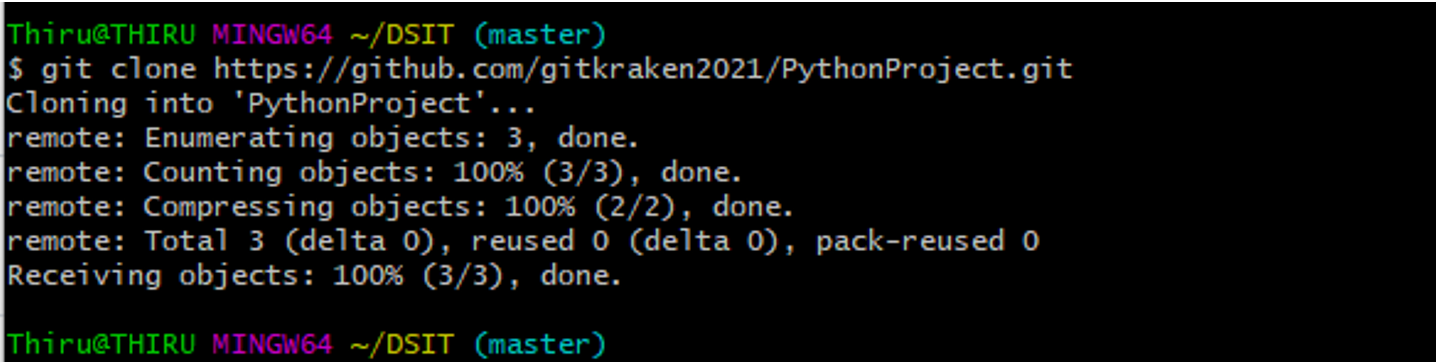
* Git clone is used to copy an existing Git repository into a new local directory
* Git clone action will create a new local directory for the repository
* Copies all the contents of the specified repository
* Create the remote tracked branches and checkout an initial branch locally
* By default, clone will create a reference to the remote repository called **origin**



* Cloning a repo allows us to make local changes to the repository before committing and pushing

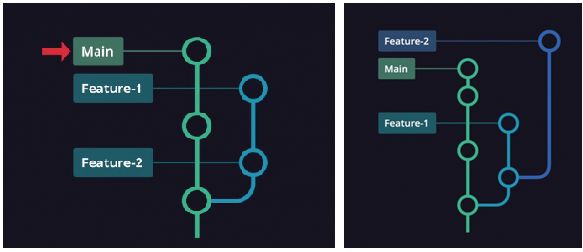
them to the remote

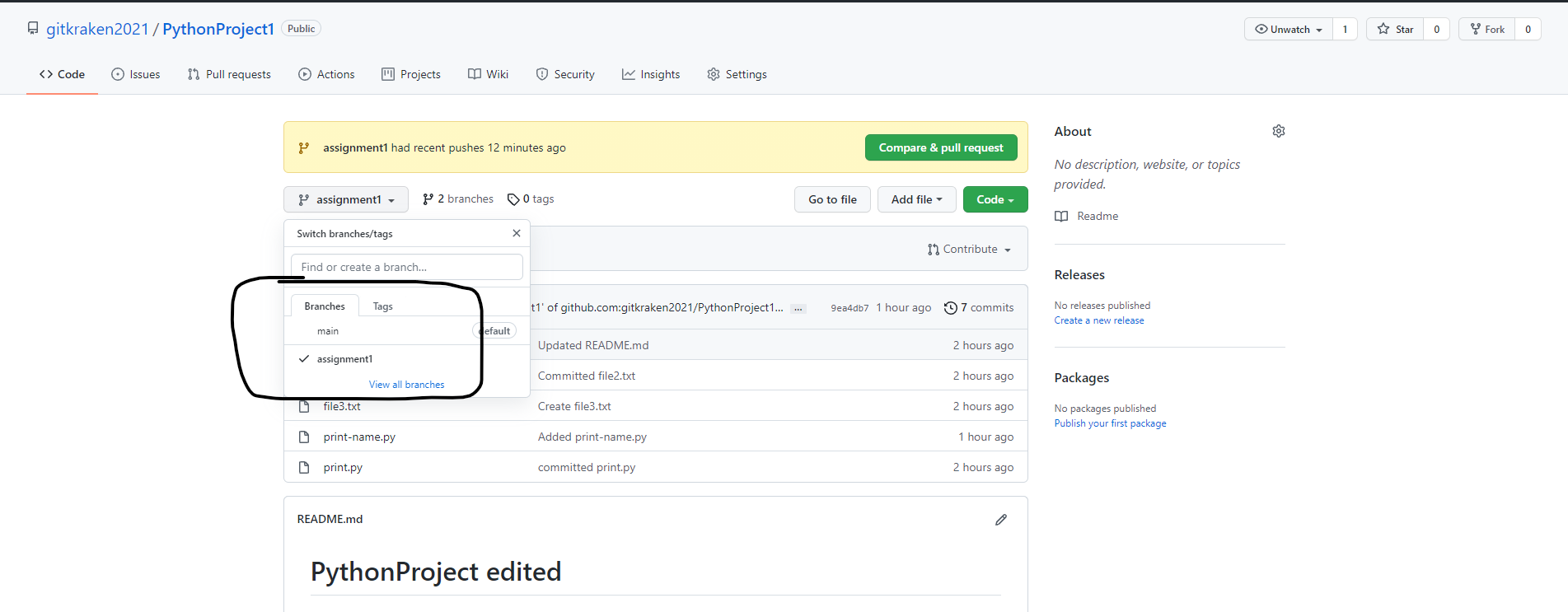
* Git clone <repository-url>



**Git Branch**

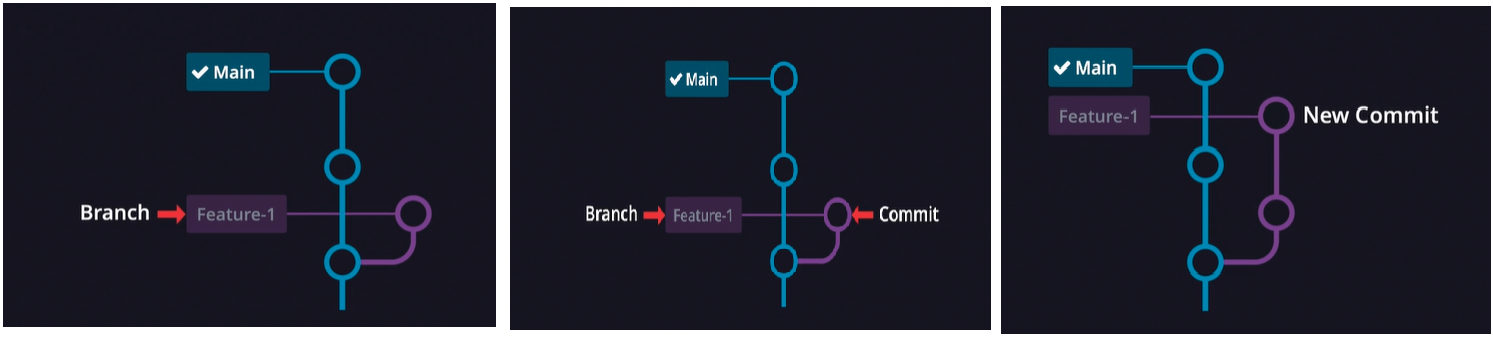
* Branch is simply a pointer to one specific commit
* The branch pointer moves along with each new commit we make
* And only diverges in the graph if a commit is made on a common ancestor commit

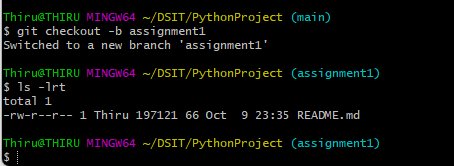


****

**Git Checkout**

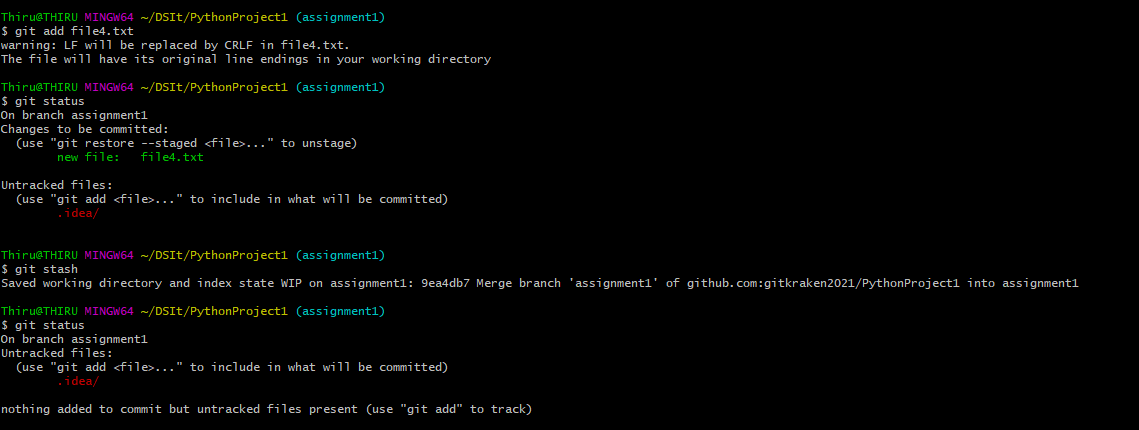
* Tells Git to which branch or commit we want our changes applied



****

**Git Stash**

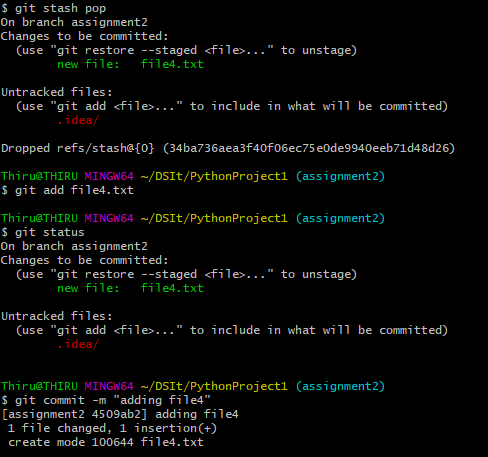
* Allows us to save our file changes for later
* Stashing is used for temporary storage of the change files and its not a substitute for committing changes

****

* Checkout to different branch

****

* Git stash pop

****

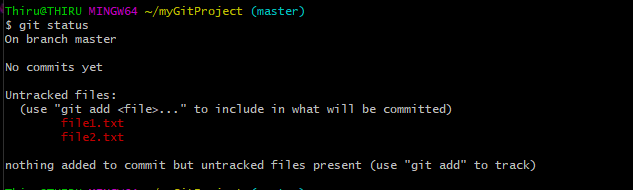
**3 Stages of Git Workflow**

* Making Changes
* Staging Changes
* Committing Changes

****

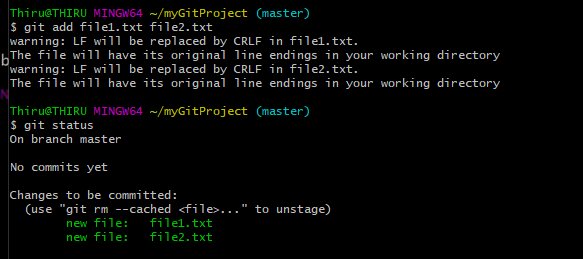
**Git Status**

* Shows what files have been added/deleted/modified/renamed
* Shows which files are staged for commit
* Shows status of current branch
* Shows which files are untracked

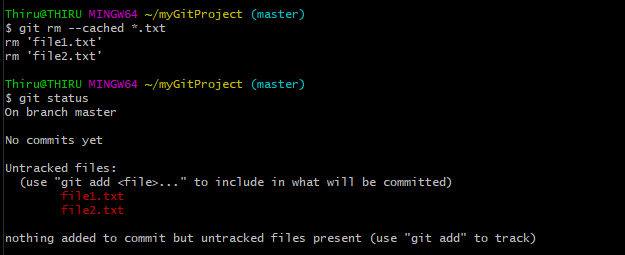
****

**Adding files to Staging**

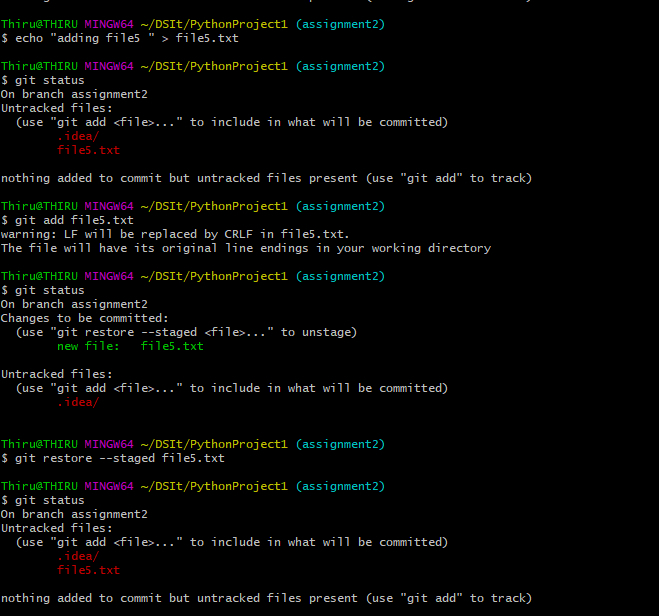
* Can add files with git add file names separated by space or by using regular expressions or . (dot)

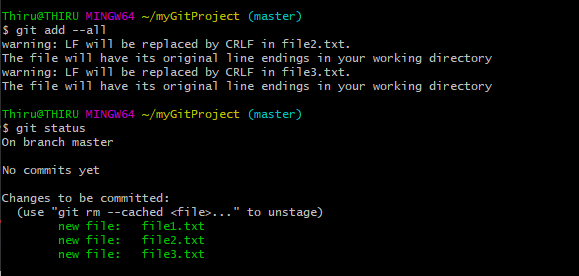
****

**Git rm --cached**

****

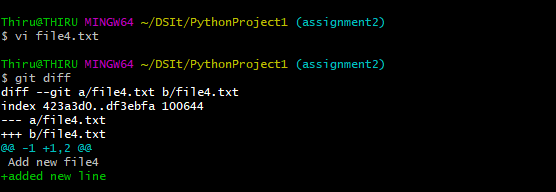
**Git restore –staged**

****

**Adding only new/modified files to Staging**

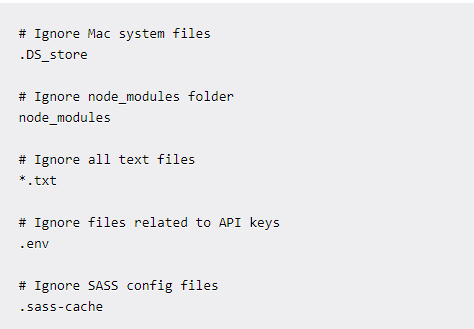
**Git Diff**

* Compares the differences between two commits
* Show what files were added, deleted or modified
* Added lines of text are commonly highlighted in green and denoted with a + sign
* Deleted lines of text are commonly highlighted in red or denoted with a – sign
* Useful to review changes while staging
* Decide whether to rebase or merge

****

**.Gitignore**

* The .gitignore file is a text file that tells Git which files or folders to ignore in a project
* To create a local .gitignore file, create a text file and name it .gitignore (remember to include the . at the beginning). Then edit this file as needed. Each new line should list an additional file or folder that you want Git to ignore.
* The entries in this file can also follow a matching pattern.
* \* is used as a wildcard match
* / is used to ignore pathnames relative to the .gitignore file
* # is used to add comments to a .gitignore file

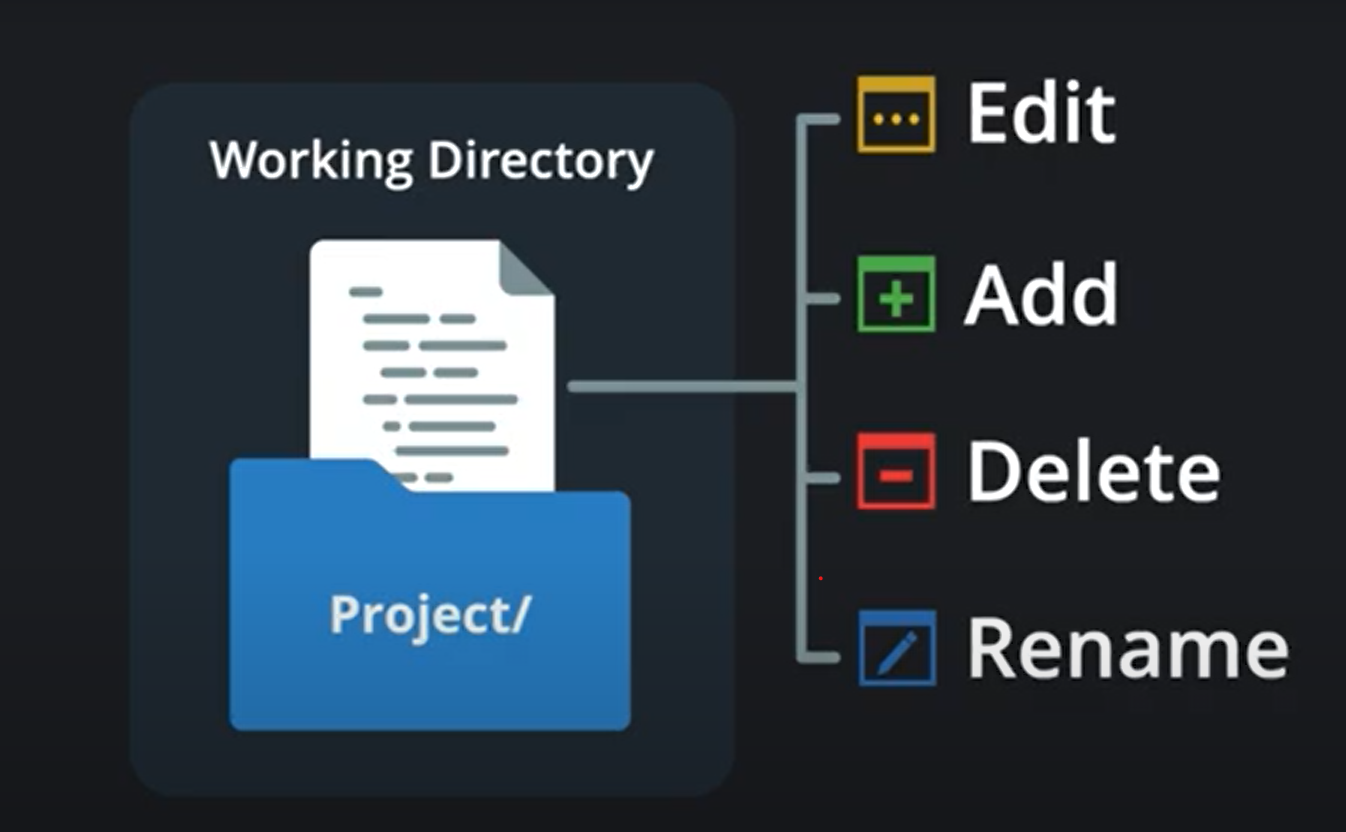
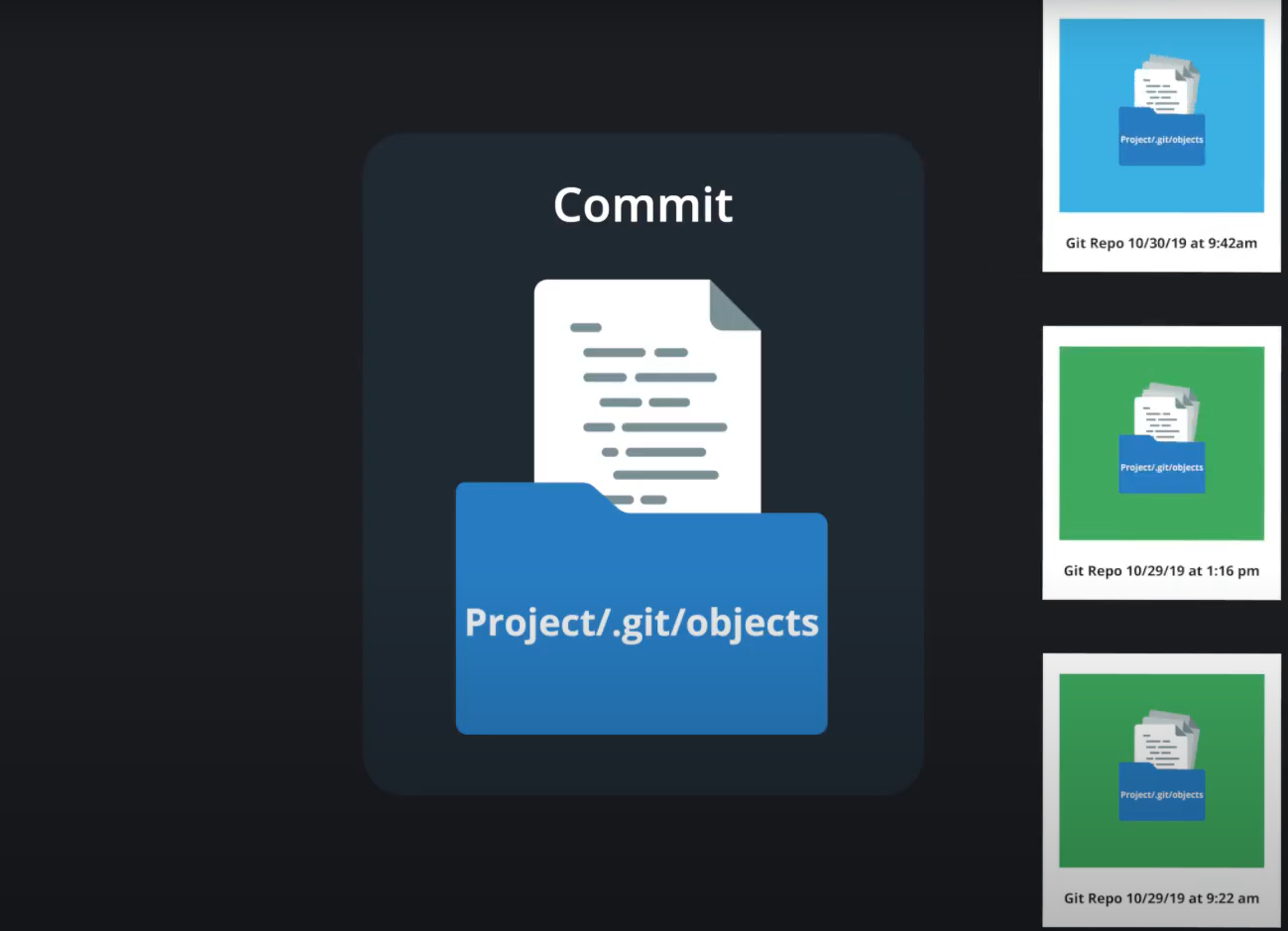


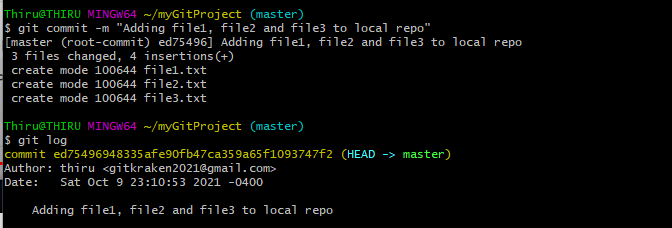


All of your Git repositories will ignore the files and folders listed in the global .gitignore file.

**Git Commit**

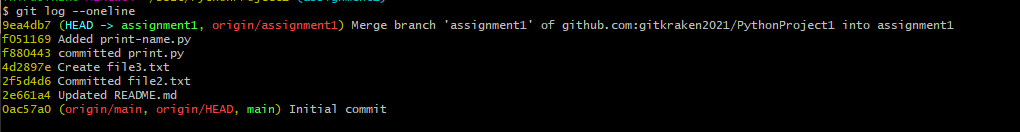
* A commit is a snapshot of our Git Repository at one point in time
* A file when edited, added, deleted or renamed is not stored in the Repo
* Once changes are staged and saved in the repository by performing a commit
* Each commit becomes a snapshot

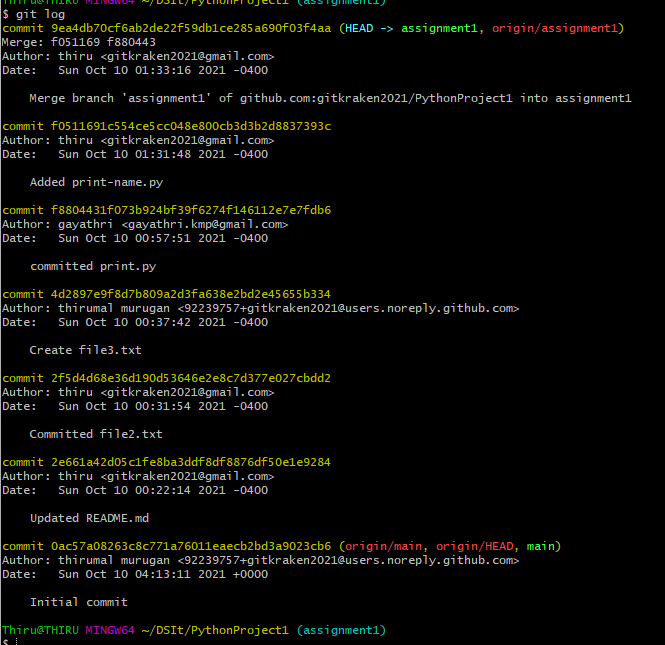
****

**Git Log**

* Shows the history of all commits starting from the recent commits to the oldest one
* It shows the commit logs in the chronological order starting from the recent commit
* Shows a 40 hexadecimal characters unique identifier for each commit
* Maintains the track history of date timestamp and the author who commits and when the commit is made and the description of the commit message
* Shows one lines of summary

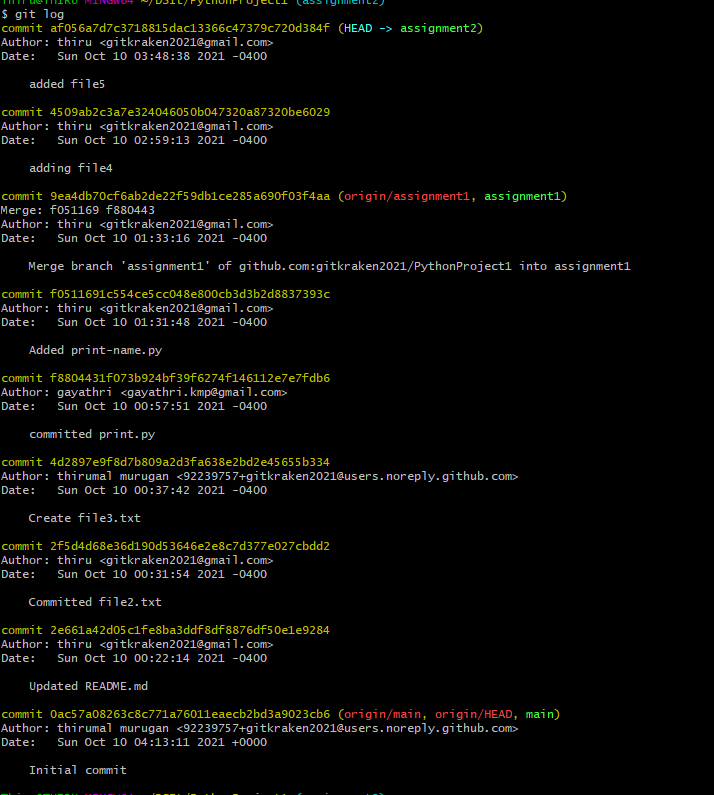


* Git log –stat shows the statistics of the file addition/deletion/modification
  + Shows the relative change with +/- sign
* Git log –patch shows the difference between the subsequent commits
* Git log graph shows the graphical representation of the commit summary
* 

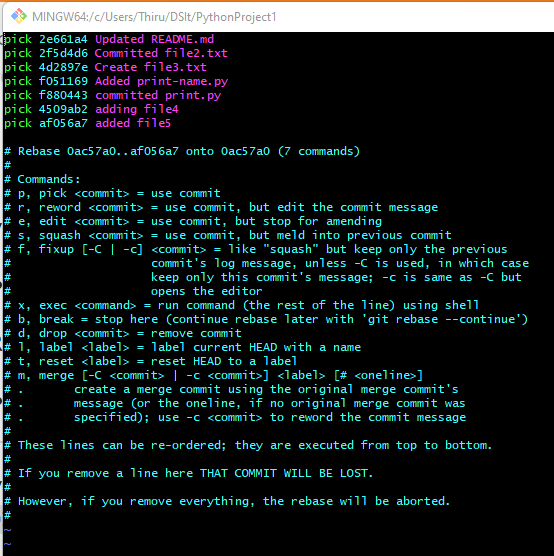
****

**Git Squash**

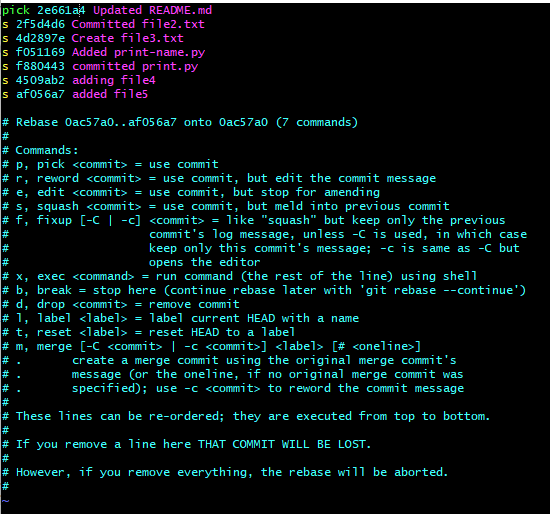
* Squashing is a way to rewrite our commit history
* This action helps to clean up and simplify our commit history before sharing our work with team members

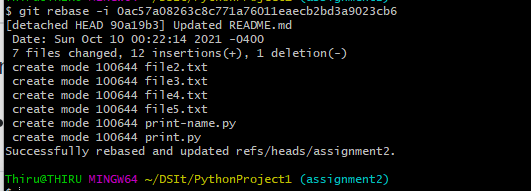


* Git Squash with Rebase



Squashing the unwanted commits in vim



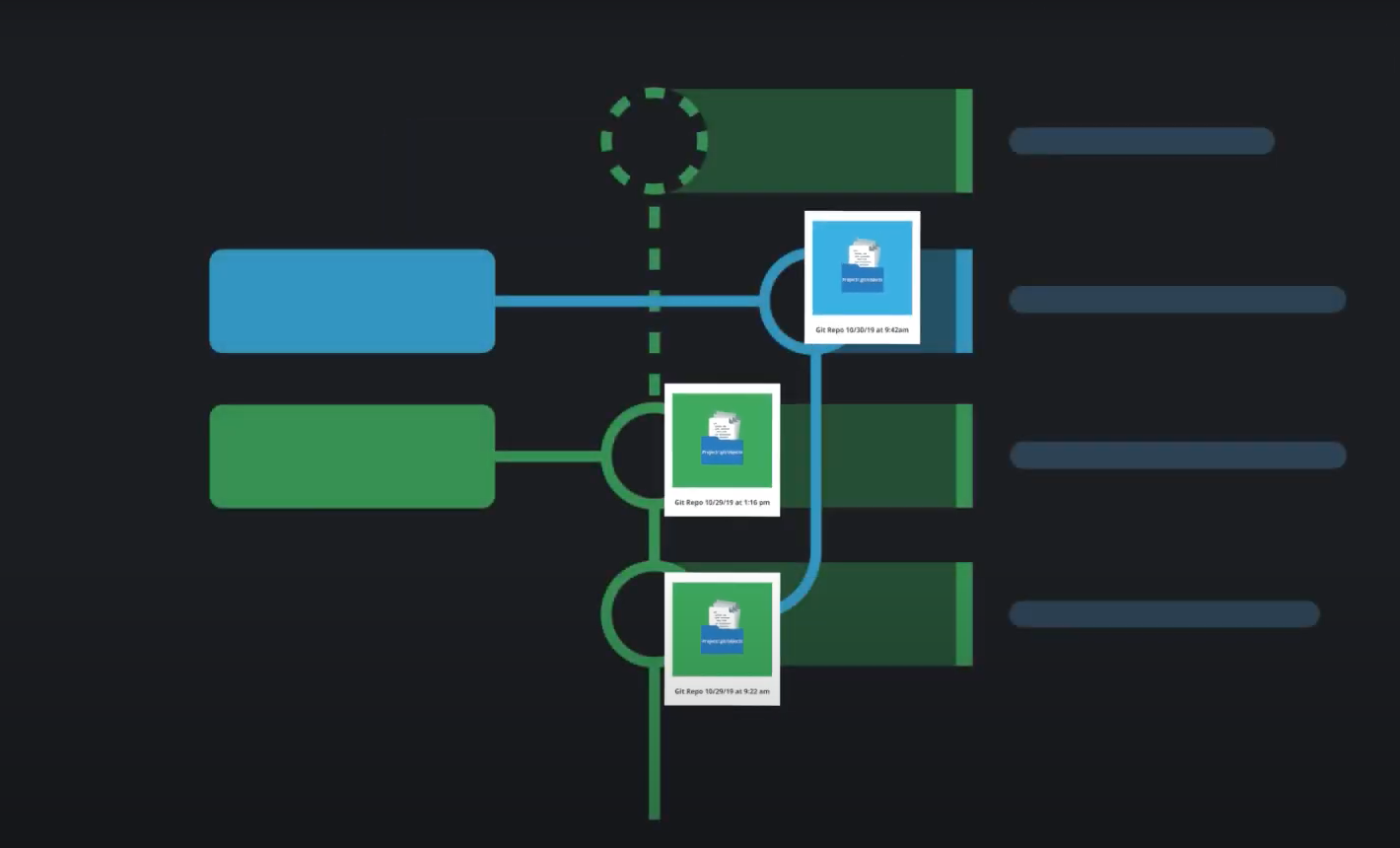
****

**Git Reset**

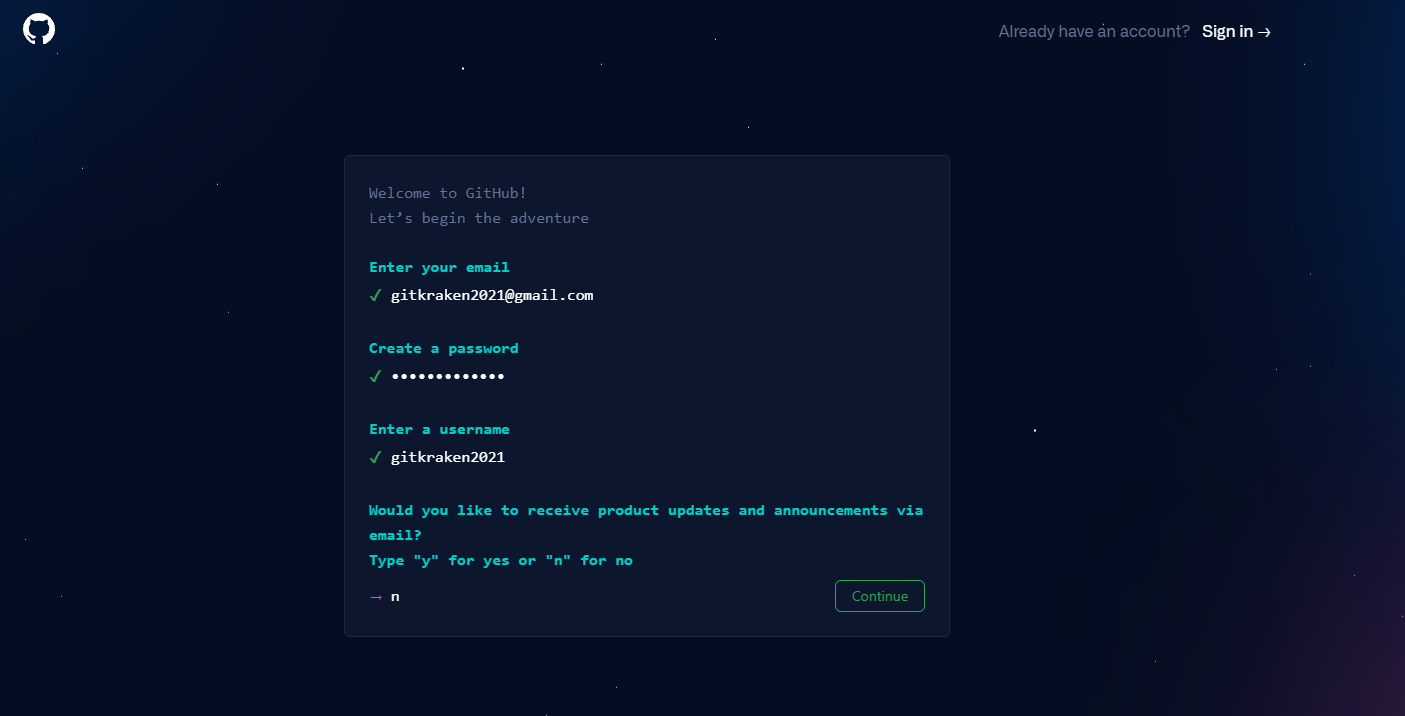
* Reset current HEAD to the specified state
* git reset --hard HEAD~1

**Git Clean**

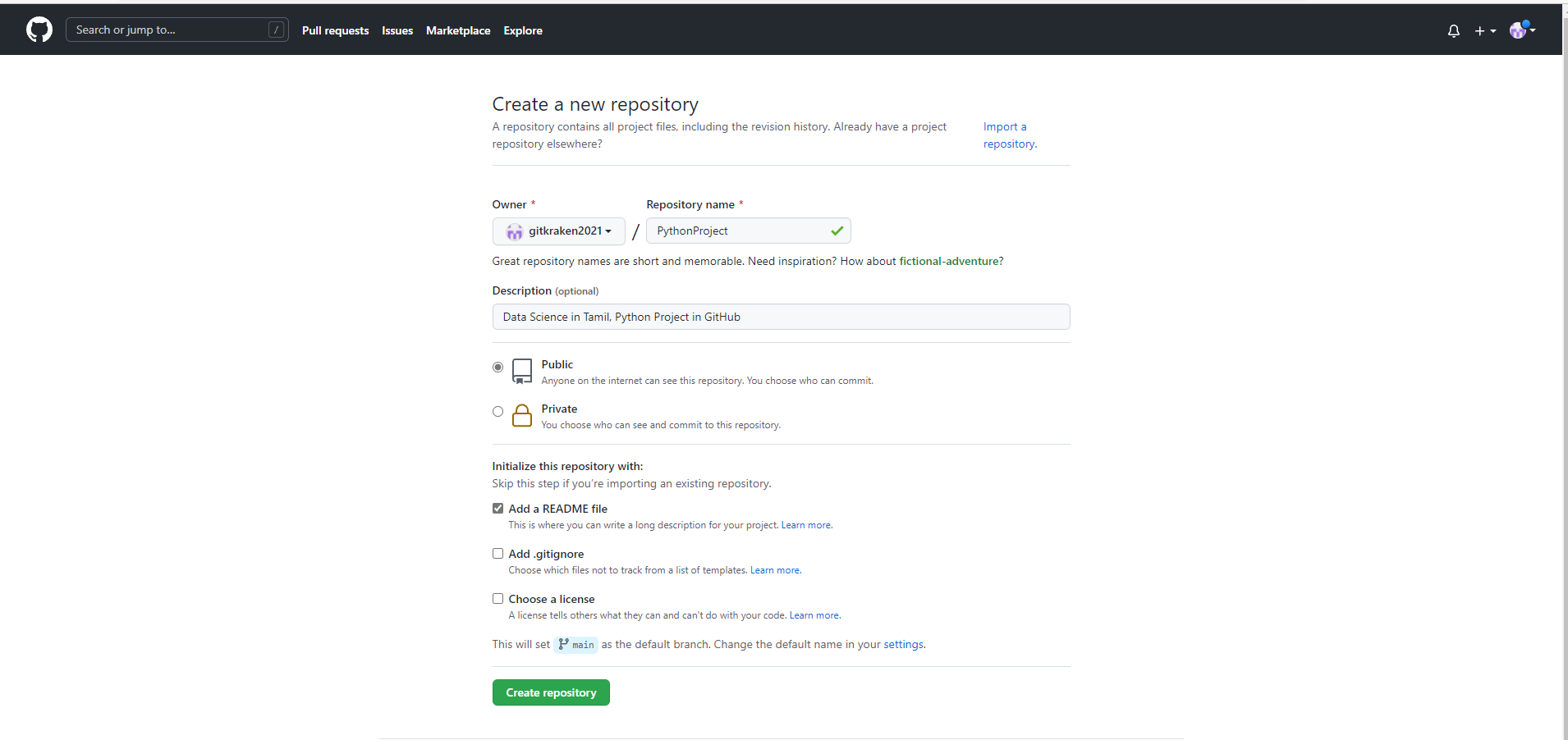
* Remove untracked files from the working tree
* **git clean** [-d][-f][-i][-n][-q][-e <pattern>] [-x | -X] [--]<path>…
* Remove untracked files from the working tree
* git clean -d -i #displays the untracked files and directories to be removed

**Cumulatively formed Repo History represented as a Graph**

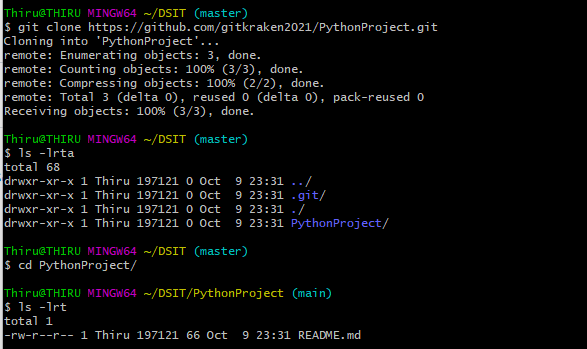
**Creating GitHub Account**

****

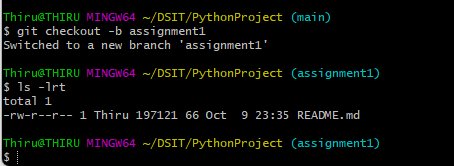
**Creating a new remote repository in GitHub**

****

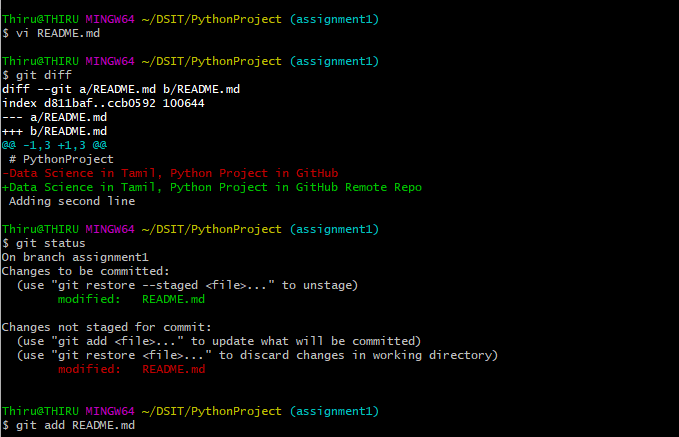
**Cloning the remote repository in local**

****

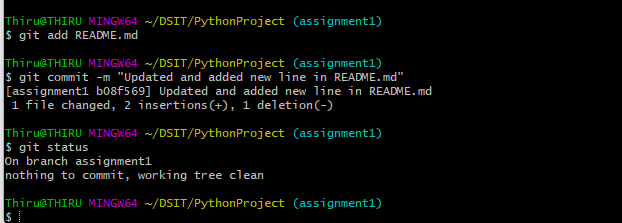
**Creating local Branches**

****

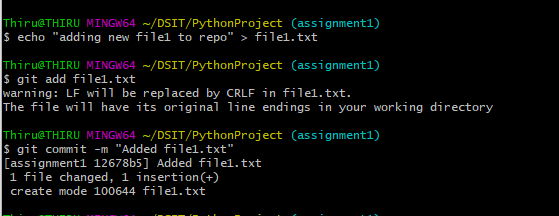
**Staging file changes in local repository**

****

**Committing on Local Repo**

****

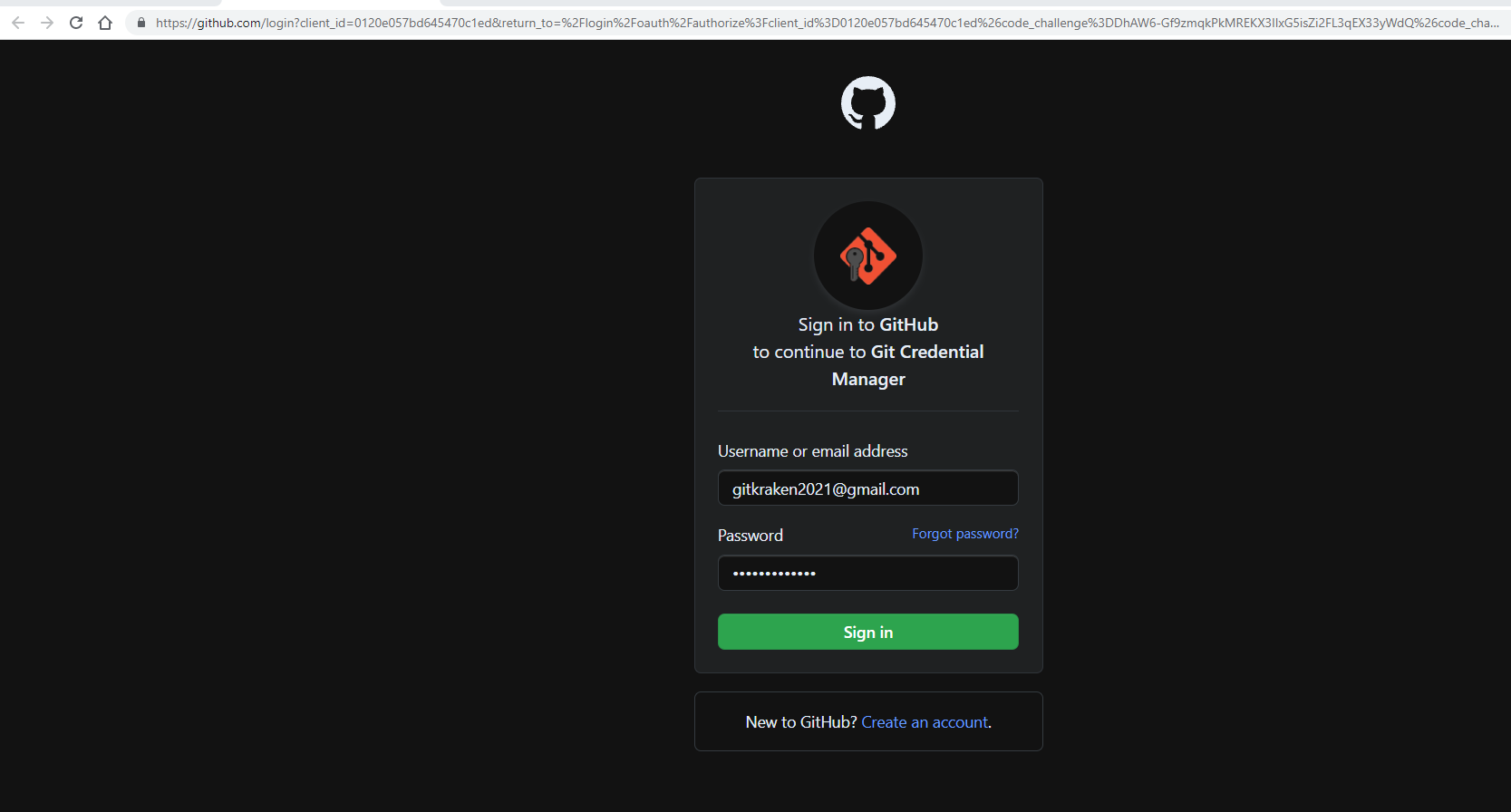
**Adding new file**

****

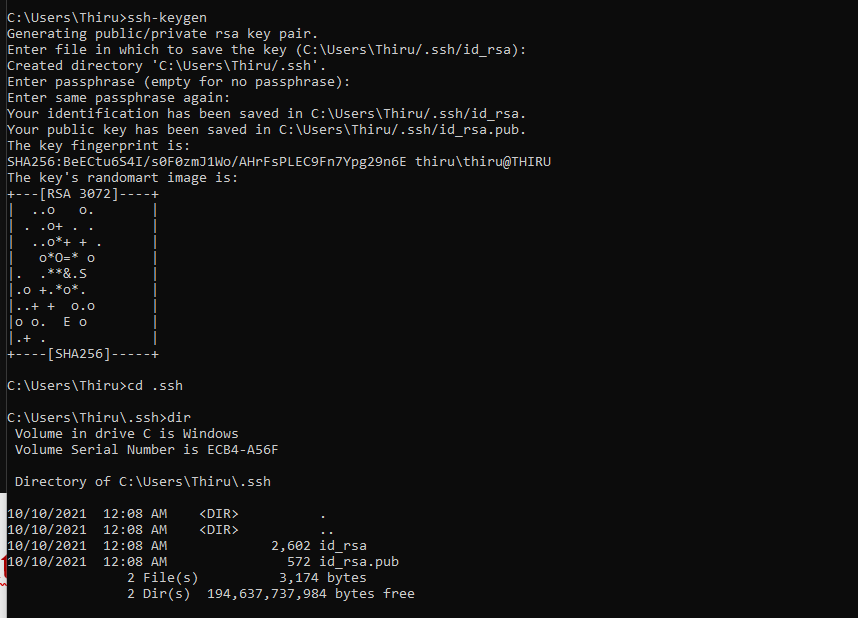
**Commit Log**

Every commit has a unique log and history

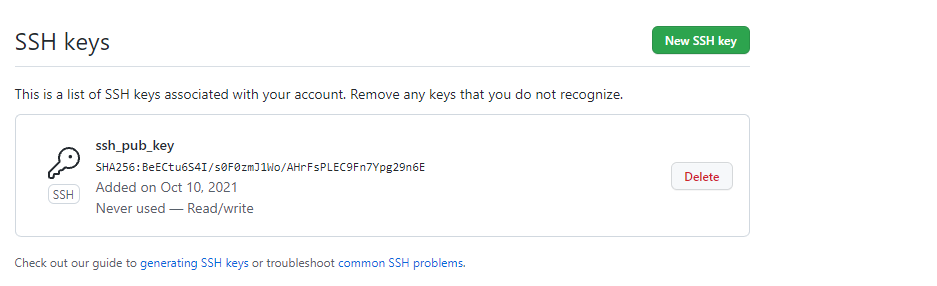
**Login & Authorizing Git Credential Manager**

****

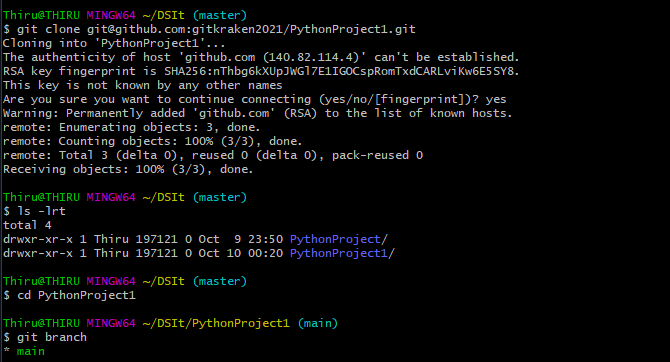
**Adding SSH keys to Github**

****

**Copy ssh-key content to Github’s SSH keys**

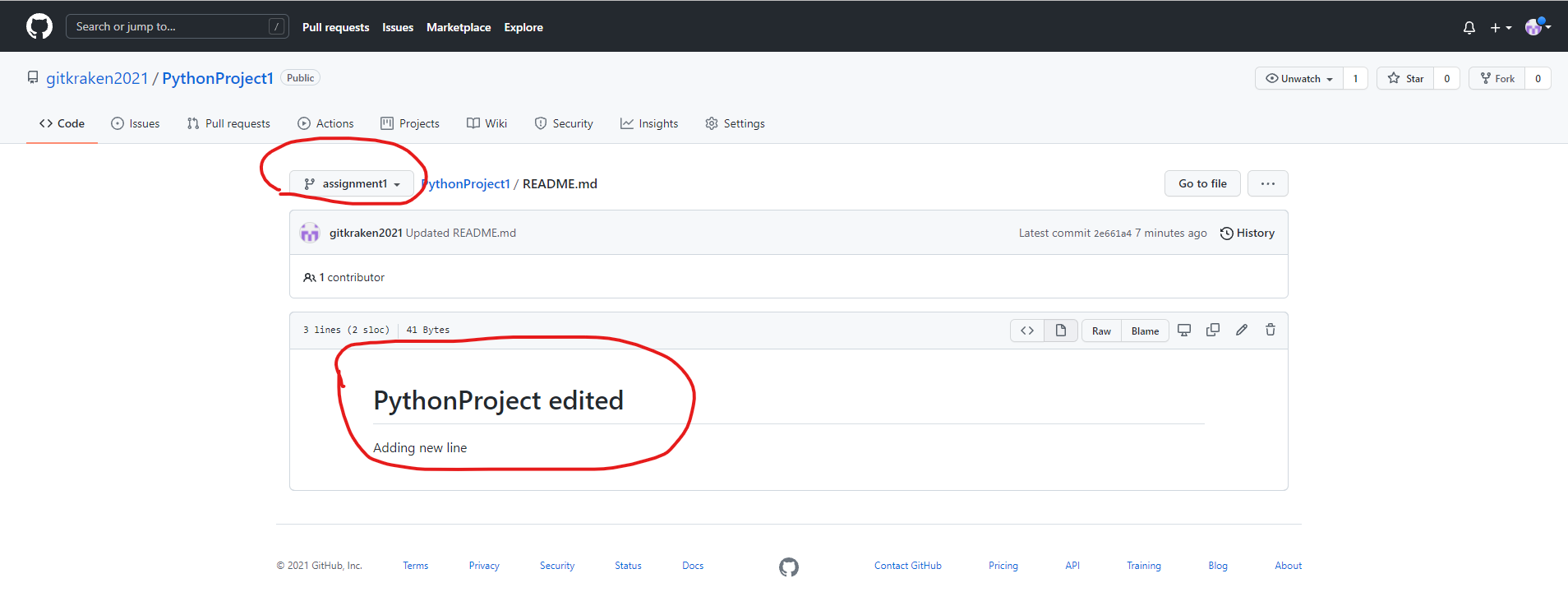
****

**Pushing local changes to remote branch**

****

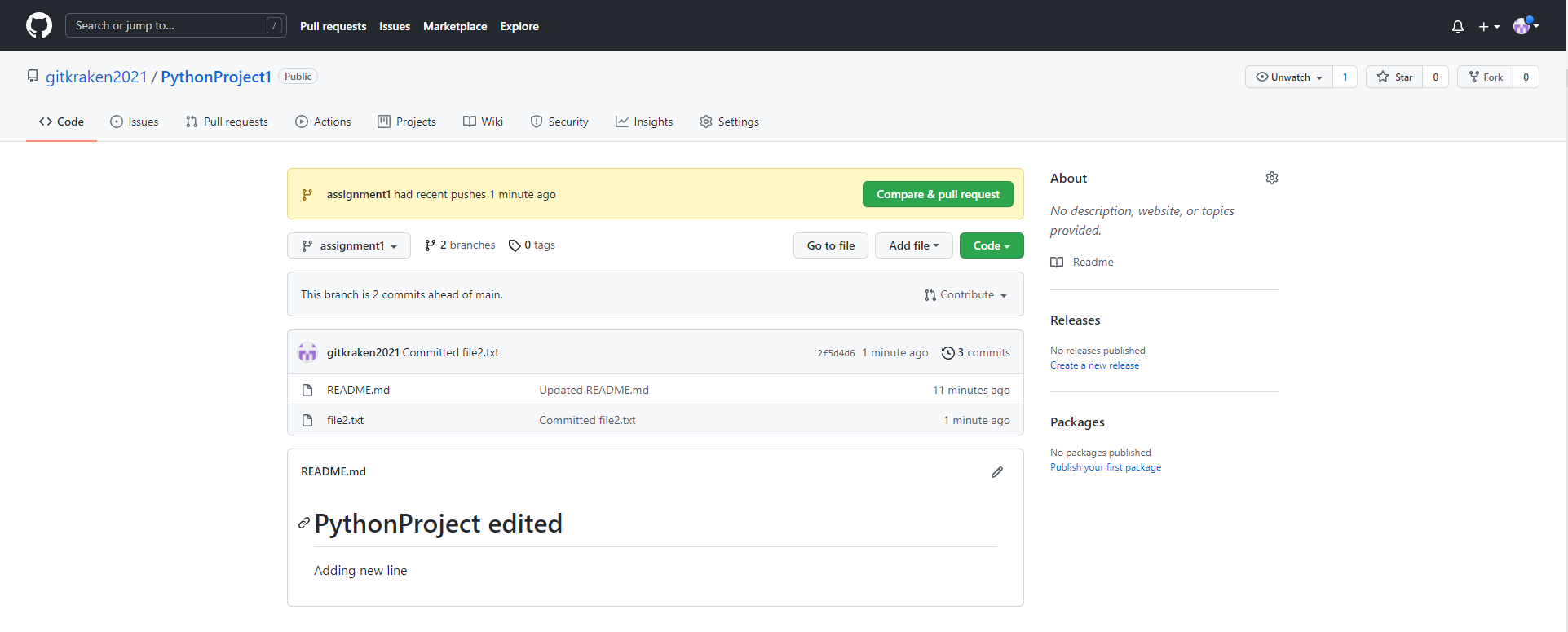
****

**Local code pushed to Remote Branch**

****

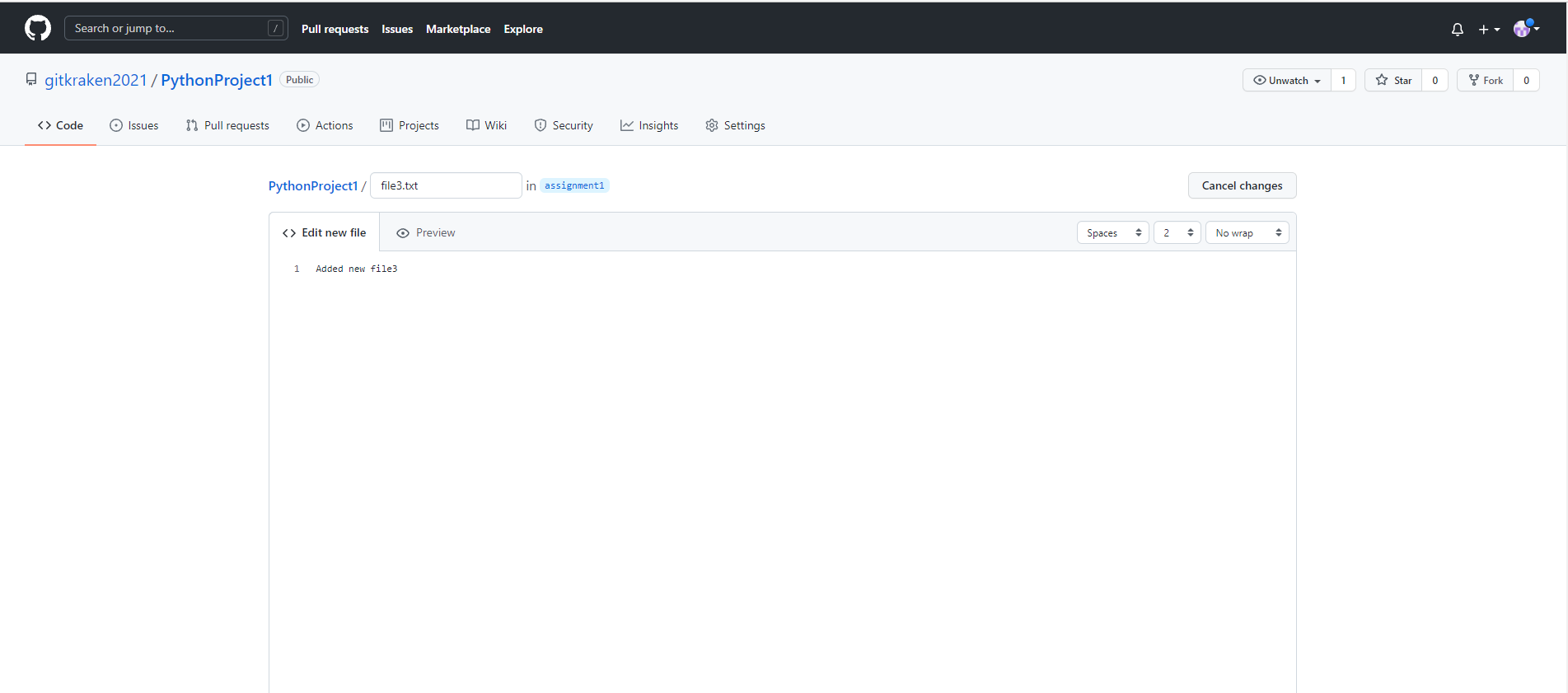
**Adding new file and pushing to Remote Branch**

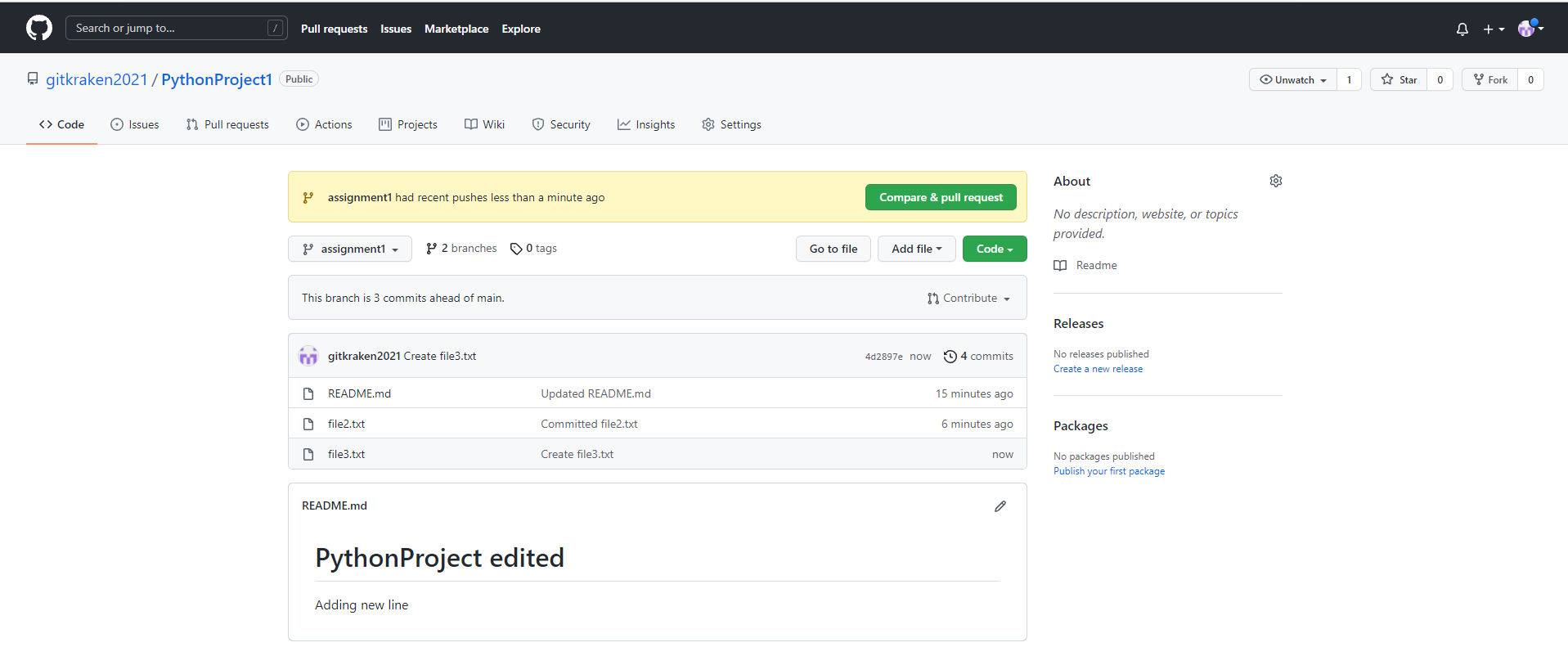
****

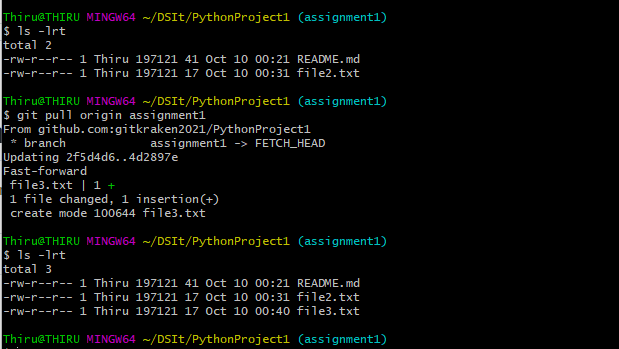
****

**Git Pull**

* Updates our local copy of the repo with the changes from the remote branch
* Git Pull = Git Fetch + Git Merge

****

****

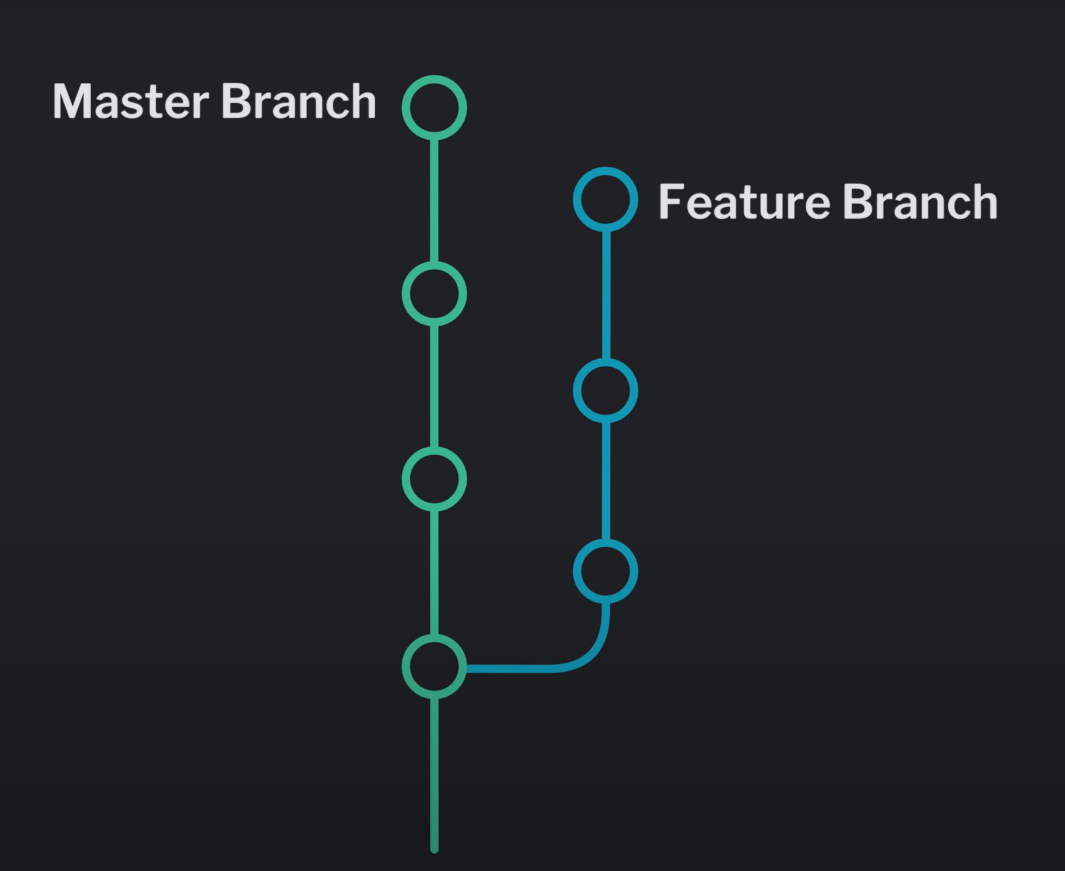
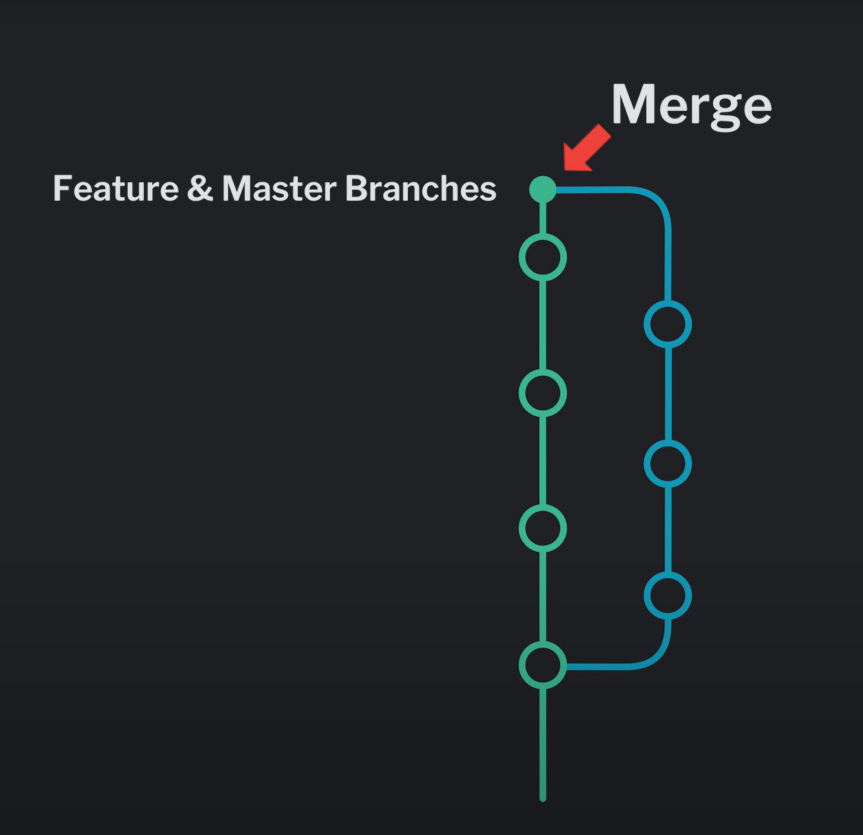
****

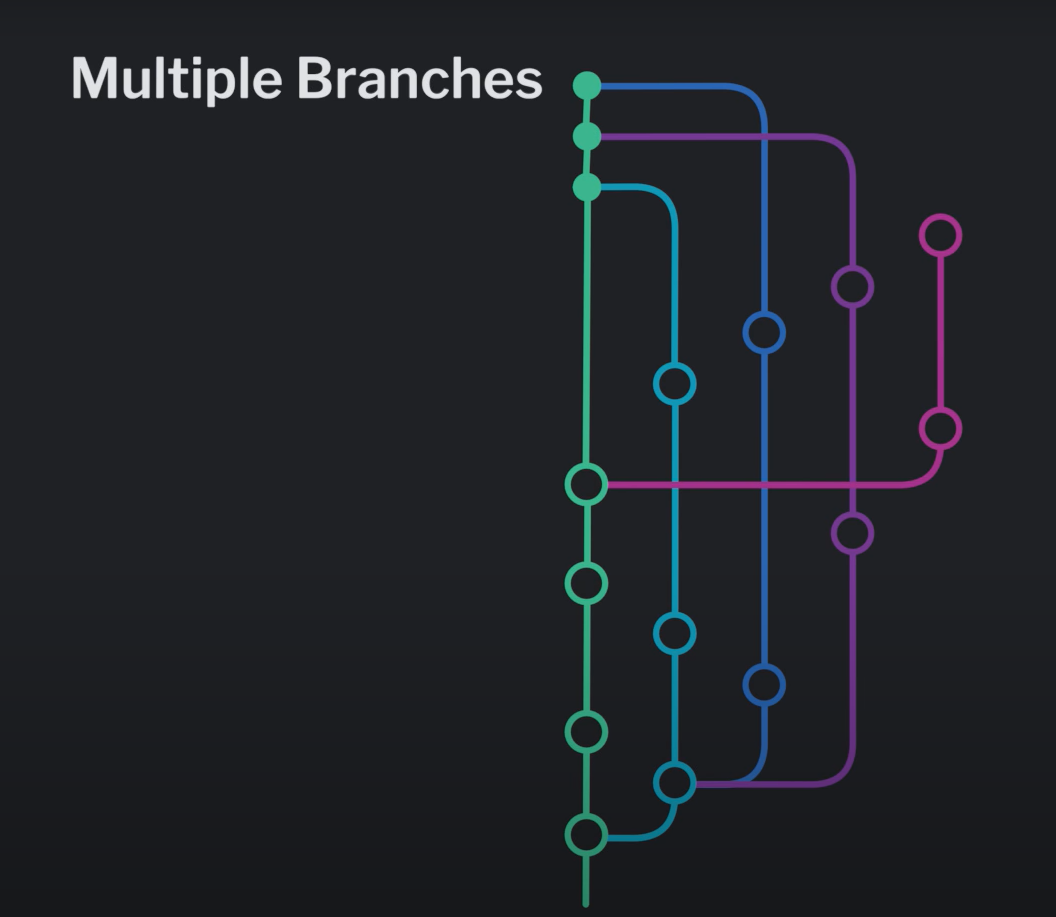
**Git Fetch**

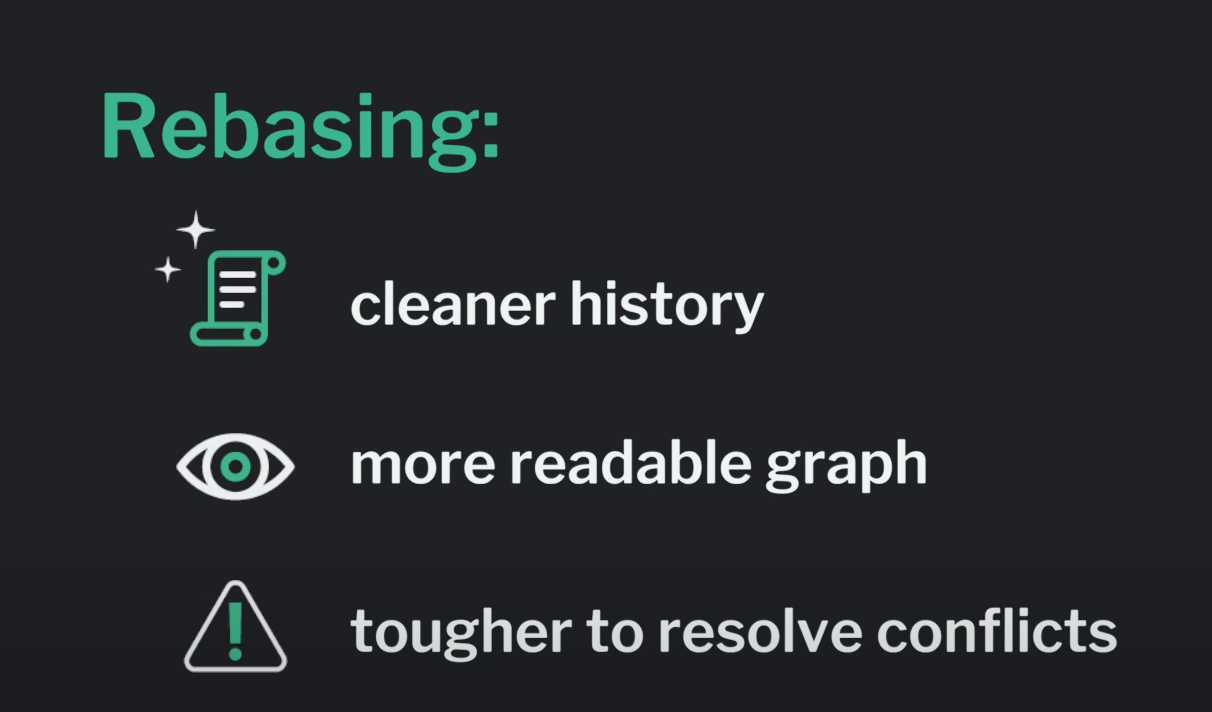
* Git fetch is a command that allows us to download objects from another repository
* All of the remote changes do not apply
* Gives more control over what’s happening in their repo
* Git Fetch is safer alternative as it pulls in all the commits from the remote but doesnot make any changes to our local files

**Git Rebase Vs Git Merge**

* Rebase is an action in Git that allows us to rewrite commits from one branch onto another branch. Essentially, Git is deleting commits from one branch and adding them to another

**** **** 



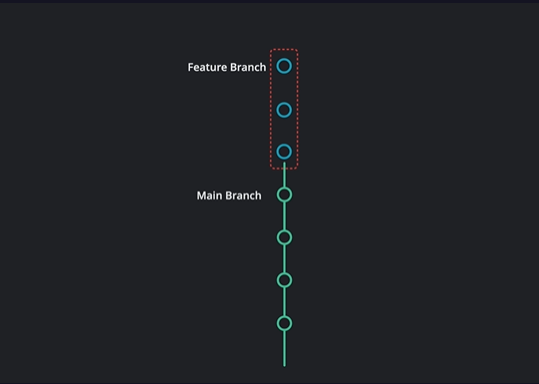
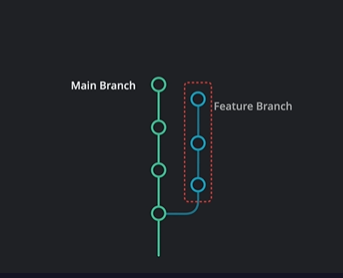
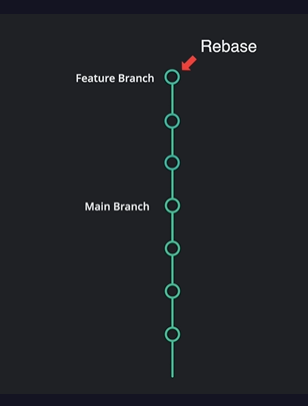


When should I use Git merge?

* Let’s say, for example, that we have a **Main** branch with changes and then we branch off into a **Feature** branch to make more changes.
* In this instance, you can combine the changes from your **Feature** branch to your **Main** branch by performing a Git merge.

When should I use Git rebase?

* If you’re dealing with numerous branches, your commit graph can become really difficult to read if you need to merge changes from various branches.
* This is where rebasing can serve as a helpful alternative to merging. Git rebase allows for a cleaner graph because it takes commits from one branch and places them onto another branch. This changes the tree structure in your graph by moving the commits and their changes onto the target branch.

**** ****

**Reference**

[Git - Book (git-scm.com)](https://git-scm.com/book/en/v2)

[Git - Videos (git-scm.com)](https://git-scm.com/videos)

[Git - External Links (git-scm.com)](https://git-scm.com/doc/ext)

[Git Commands Cheat Sheet | Learn Git (gitkraken.com)](https://www.gitkraken.com/learn/git/commands)

[Learn Git Cheat Sheet | Free Download (gitkraken.com)](https://www.gitkraken.com/pdfs/git-basics-cheat-sheet)