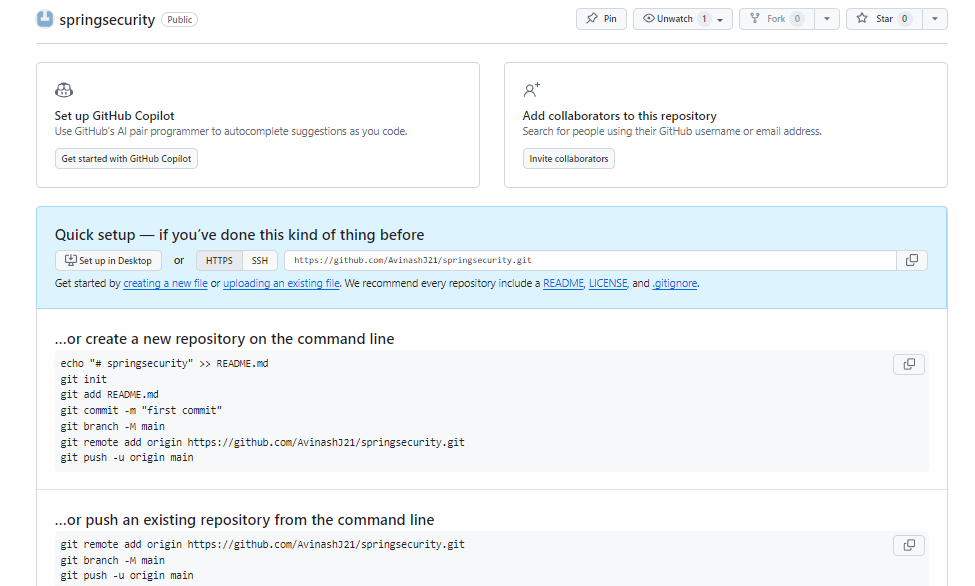
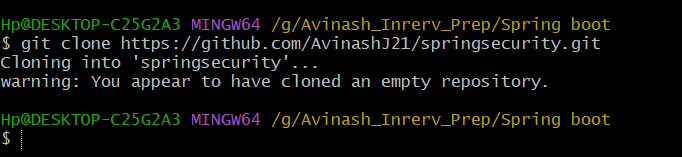
**GIT learning**

**Once you create GIT repository on git hub. You can clone the repository on your local machine. The home page of GIT repository will be provided with few of the read me commands page as below**

****

**Now download git bash on the local and goto particular folder and try to clone the repository**

****

[**https://drive.google.com/drive/folders/1ytcycUBYgnDLy1L-4bAPeSNAA\_l6OFSe**](https://drive.google.com/drive/folders/1ytcycUBYgnDLy1L-4bAPeSNAA_l6OFSe)

**Version control is process of versioning the projects or your files. Version control keep track of your changes so that we can revert back to the older changes. Also we can store centrally in repo.**

* **When working with digital files, whether is a .docx Word file or a programming .py Python file, you probably don’t finish all your work at once.**
* **In this case, the software you use to manage and track changes in these files is your version control.**

**Earlier before Version control we used to do versioning on the local system as below**

* **You could create your own version control system, such as using multiple files:**
  + **My\_Paper.docx**
  + **My\_Paper\_v2.docx**
  + **My\_Paper\_Final.docx**
  + **My\_Paper\_Actual\_Final.docx**
* **But this creates mess of maintaining instead we can rely on versioning control system**
* **Version Control**
  + **Alternatively, you could decide just to keep updating the same file over and over again.**
  + **But what happens if you want to go back to a historical version, perhaps because of a bug in your newest version?**
  + **What if you want to work with others on the same code?**
* **Version Control**
  + **Clearly we need a more robust system that can allow us to work on the same file, yet retain information about previous versions.**
  + **A version control system (VCS) allows us to track changes, undo changes, compare versions, work with others, and more!**
* **Git itself is one example of a VCS, and while there are others (subversion, mercurial, etc…), git is by far the most popular VCS in the world.**
* **Fortunately for us git is free and open-source and we can download it to our own computer.**
* **Version Control**
  + ***What is GitHub? (or GitLab?)***
    - **You can use git completely free locally, and use it to track changes on your local files on your own computer.**
    - **However if you want to host your code on the internet, its typically easier to use a hosting service like GitHub.**
* ***What is GitHub? (or GitLab?)***
  + **GitHub (owned by Microsoft) integrates easily with git and allows you to have versions of your code “live” on the internet for you and others to access, update, and change from any machine.**
* ***GitHub vs. git***
  + **GitHub is not the same thing as “git”.**
    - **Git:**
      * **Open source VCS software.**
    - **GitHub:**
      * **Company that operates a service for hosting files on the internet that are managed using git.**
* **Version Control**
  + **A VCS such as git allows you to add “save points” to your work, create changes, update existing code, undo changes, create branches for others to work on, and merge work together.**

**Command prompt commands**

**cd dir : to list all the directories**

**If you want to move to some location say Desktop**

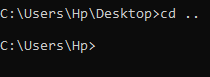
**cd Desktop**

**To clear command prompt**

**cls**

**To move back from current directory**

**cd..**

****

**To create a folder**

**mkdir folder\_name**

**To move to the different drive**

**E:**

* **Also just a quick note, you will see it stylized as git or Git.**
* **Git typically refers to the entire project, while git is the actual program used.**
* **Git is the open source software that actually manages the git commands as a VCS.**
* **We will call it at the command line with commands such as: git push**
* **GitHub is the online hosting provider, that can act as a machine connected to our local machine via the internet and host code in a repository for us.**
* **GitHub also provides many other features, especially graphical interfaces of many git features.**
* **For example you can easily merge issues or view commits on the GitHub website, rather using the command line locally.**
* **An online hosted repository also provides an ideal connection point between everyone working on the same project.**
* **Windows Users:**
  + **Our *HIGHLY recommend* text editor for this course is VS Code:**
    - [**https://code.visualstudio.com/**](https://code.visualstudio.com/)
  + **Why *HIGHLY* recommended?**
    - **Windows + VS Code + GitHub**
    - **Upon installing git you will be asked to select a default editor, you’ll need VS Code installed to select it as default.**
* **Windows Users:**
  + **Next we’ll download git, go to:**
    - **https://git-scm.com/**

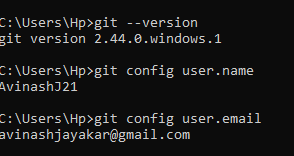
**Also download github desktop for windows once done then configure the account.**

**To check the configuration go to command prompt and type below commands**

**git –version**

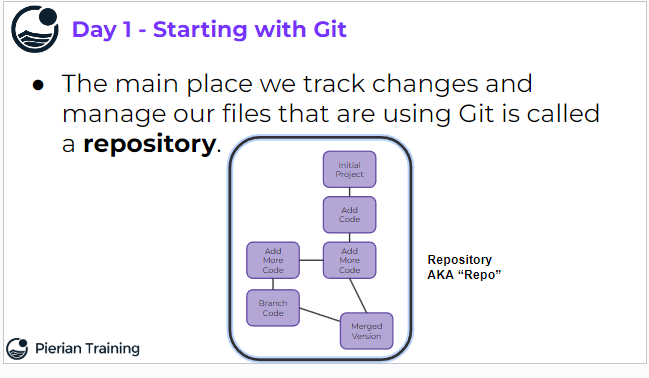
**git config user.name**

**git config user.email**

****

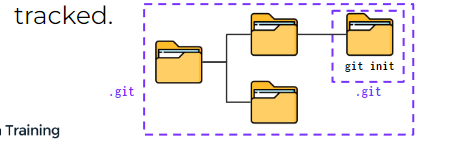
**If we want to update to another account**

* **git config --global user.name “user”**
* **git config --global user.email “email”**

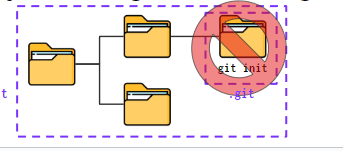
****

* **How can we create a Git Repository?**
  + **git init**
    - **This command initializes a Git Repository on your local machine.**
    - **You only need to run this command once per project.**
  + **git status**
    - **This command will report back the status of your Git repository.**
* **How can we create a Git Repository?**
  + **Upon creating a repository with git init you will create a hidden .git file.**
  + **The .git file is a hidden file that manages the versioning of the files inside the Git repository.**
* **Git inside a Folder/Directory:**
  + **Upon creating a Git Repository, all the folders/directories inside the top level Git Repository will also be part of that Repository, meaning all the changes are tracked.**

**Its bad practice to create repository within another repository**

****

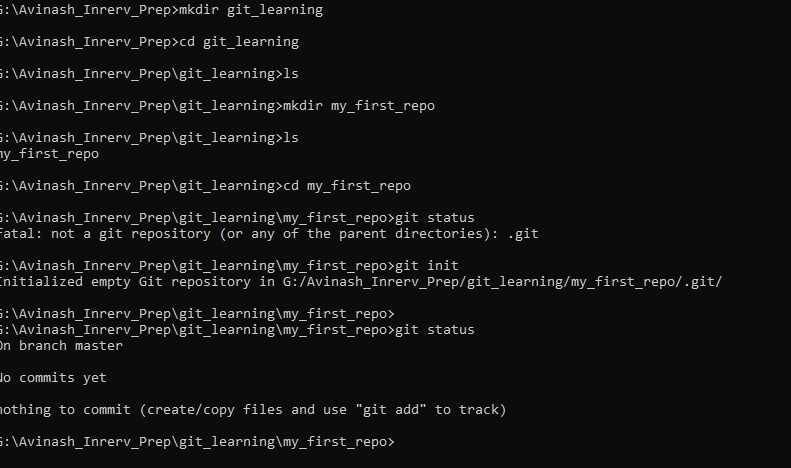
**Avoid it**

****

**Lets create local repository**

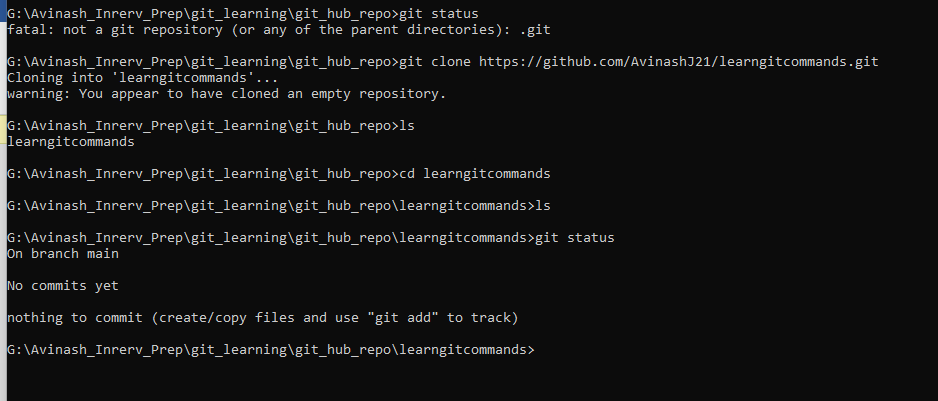
**git init is used to create repository it actually creates a .git file which keeps track of all the changes**

**by default this repo will be master branch**

****

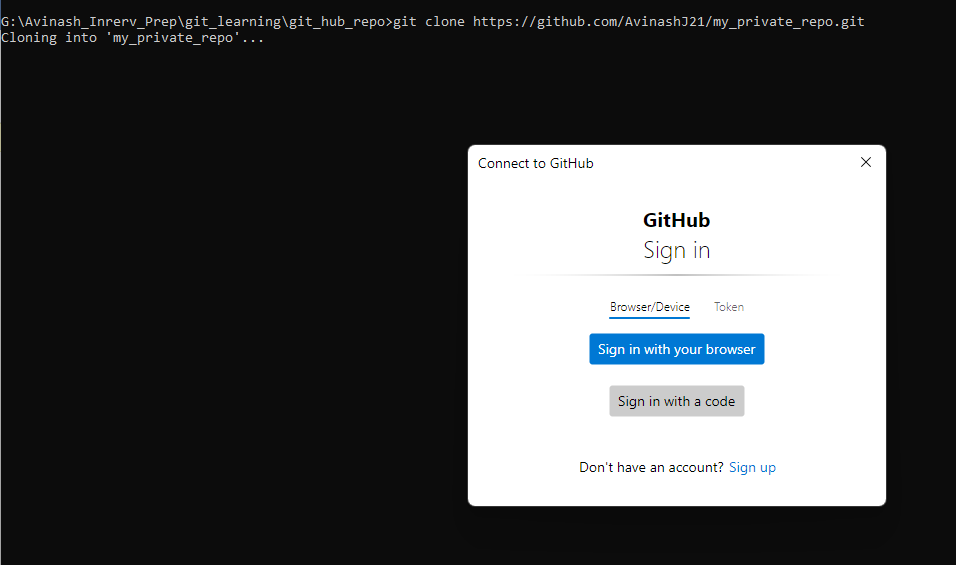
**To clone repository from git hub**

**git clone** [**https://github.com/AvinashJ21/learngitcommands.git**](https://github.com/AvinashJ21/learngitcommands.git)

****

**For public repo its pretty straight forward just clone the repo on local machine. However when we want clone private repo we need to have token**

**You will get popup saying signin we can click token option in popup. So to create token we need to configure it on git hub**

****

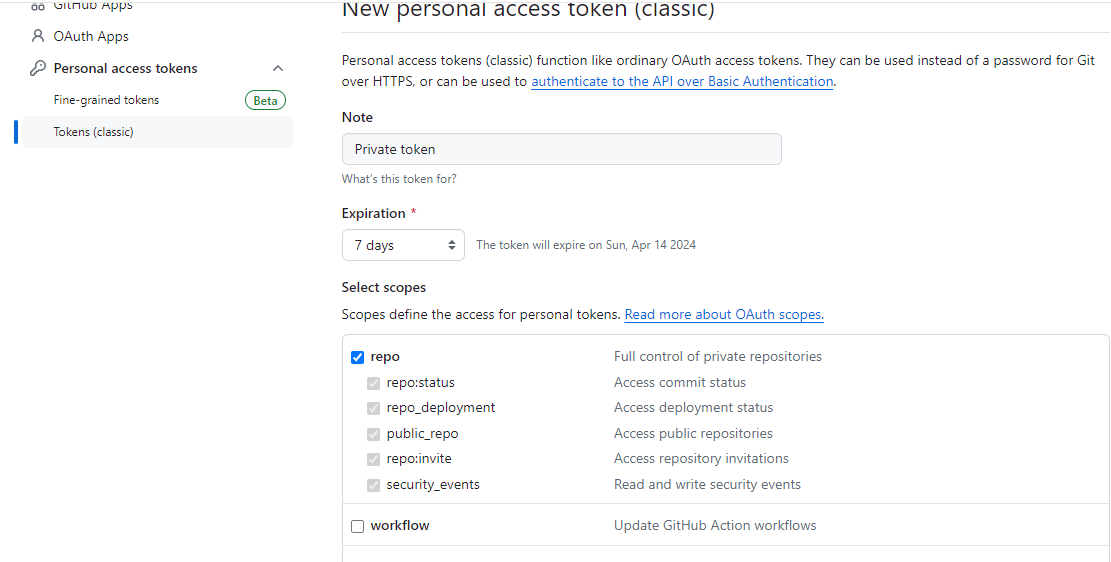
* **Option 1: Command Line:**
  + **Create Personal Access Tokens (PAT) on Github.com**
  + **When using the git clone command, reference the PAT.**
* **Option 2: GitHub Desktop Tool GUI:**
  + **Open the Github Desktop Tool**
  + **Login with GitHub Username and PW**
  + **Clone Repo via GUI**

**Through command line cloning.**

**Goto github repo click profile icon on right side then settings**

**Then go to developer settings left panel side. Then its self explainatory**

**While creating token please select repo as below**

****

**Token for our private repo**

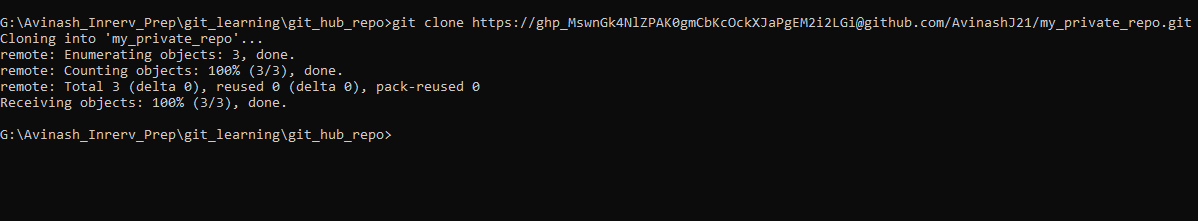
**ghp\_nr3lM12XxuQC7HgHaZohXj9xR3Rgez2xeaty**

* **Clone Syntax with PAT:**

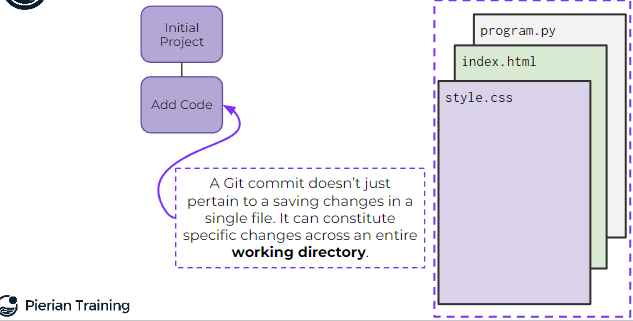
**git clone** [**https://<token>@github.com/account/repo.git**](https://%3ctoken%3e@github.com/account/repo.git)

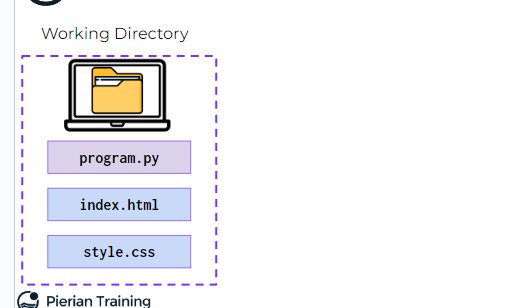
**now to clone repo with token we need to use above http format** [**https://token**](https://token)

**git clone https://ghp\_MswnGk4NlZPAK0gmCbKcOckXJaPgEM2i2LGi@github.com/AvinashJ21/my\_private\_repo.git**

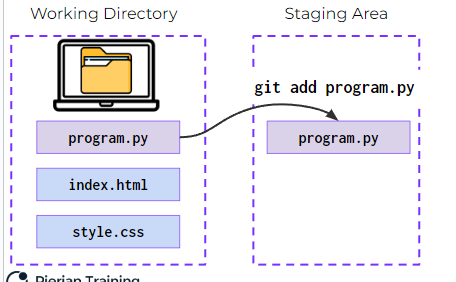
****

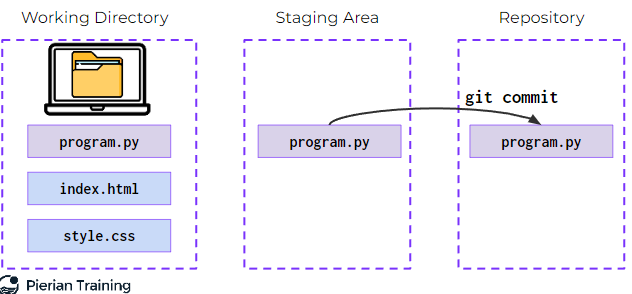
* **Day 2 Commands:**
  + **Changing code in a Repository**
    - **git add**
  + **Committing these changes**
    - **git commit**
  + **Pushing or Pulling Changes**
    - **git push and git pull**
  + **Checking Status, Logs, and Changes**
    - **git status, git log, git diff**

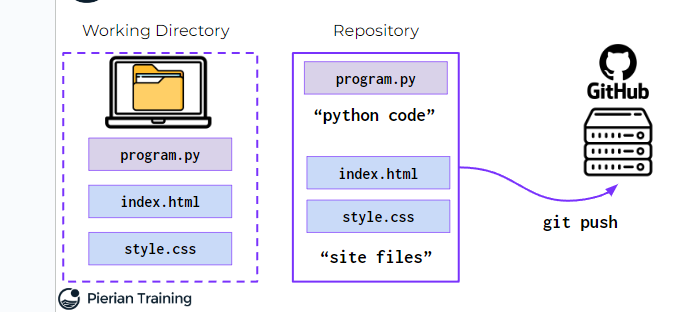
****

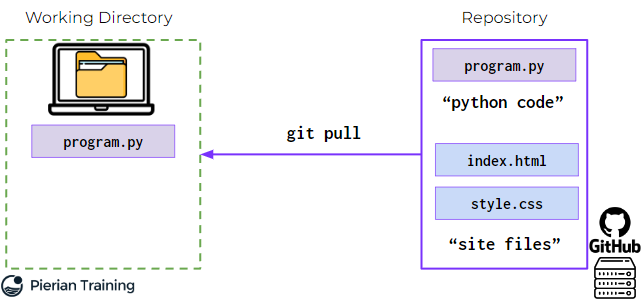
****

**Though you have added the files in your repository but GIT still doesn’t know whether these files are actually the part of repository or no for that we need to commit these files to be part of repository. This is done by adding files to the staging area and then commit and push.**

****

****

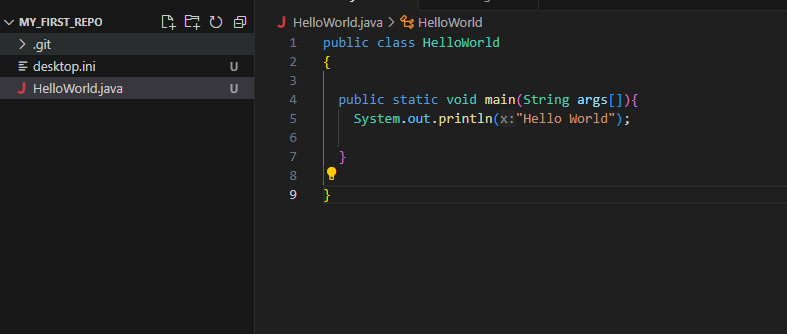
****

****

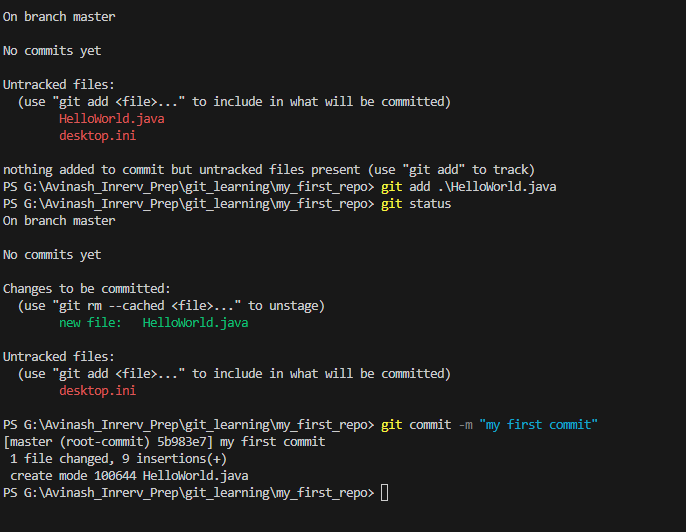
**Now create a repository in your local system and open it in VS code**

**To open in vs code goto explorer and open that path, by default git folder will be hidden. To show this folder in explorer goto settings search “files excluded” then click cross button for \*\*/.git it will open in explorer**

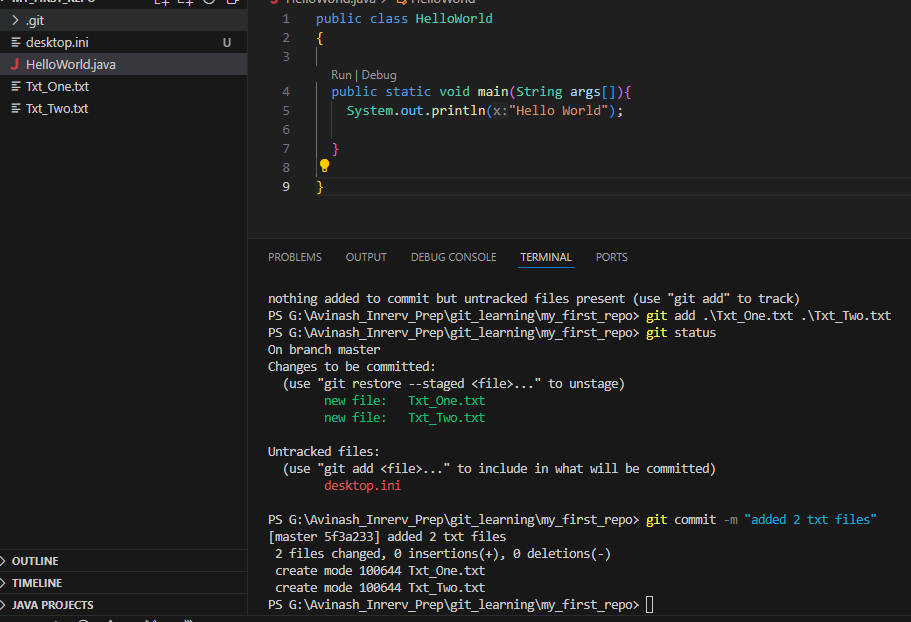
**Now lets create our first java file HelloWorld.java in repository**

****

**Now to know the GIT that this file should be the part of repository we need to first add the file in staging area and commit the fie .**

****

**You can add multiple files in staging area separate the files by space as below**

****

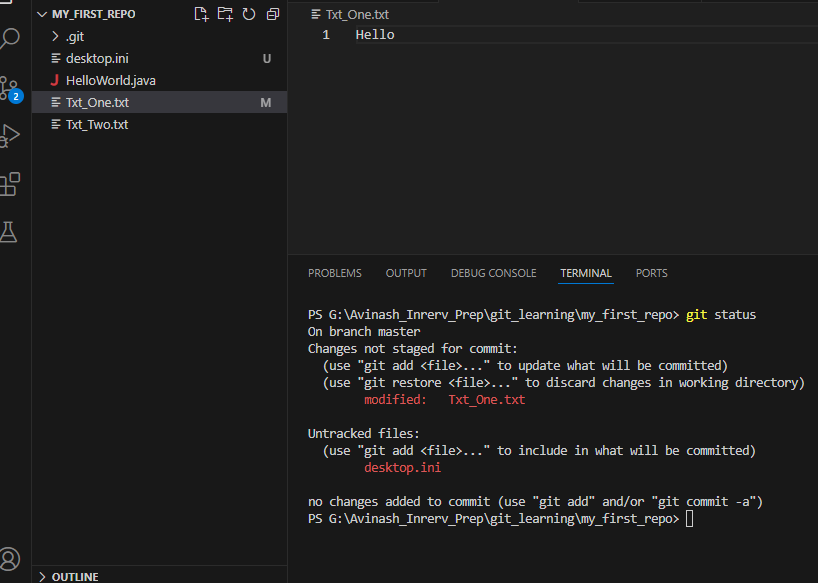
**To get add all the files**

**git add .**

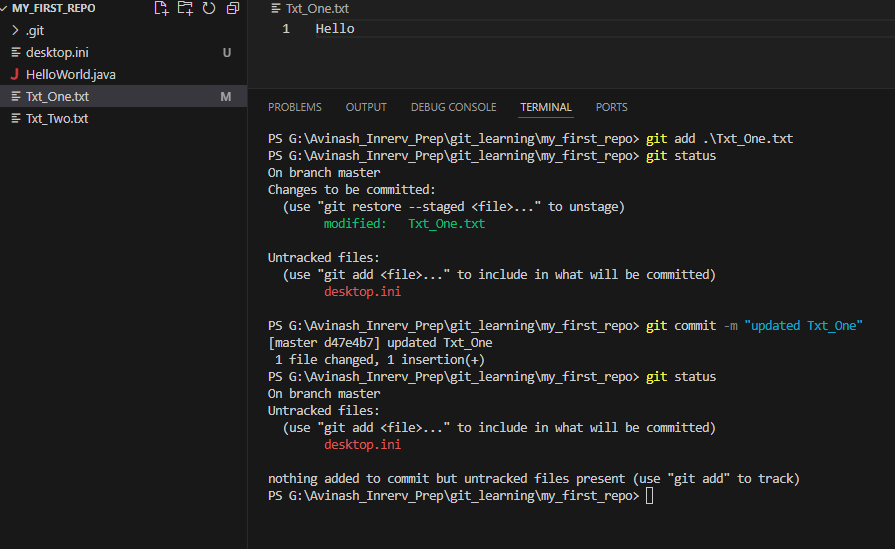
**but we need to be careful that unwanted changes may also be added in staging area**

**Modify existing file**

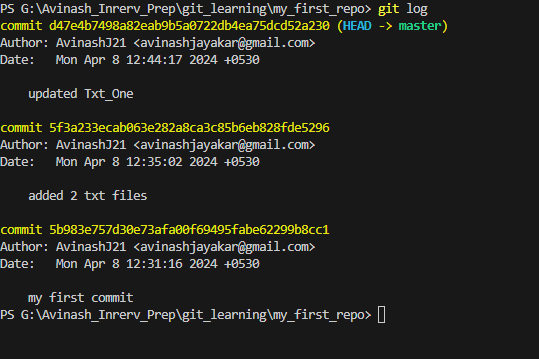
**Now I have modified one file called Txt\_One.txt git actually keeps track of it we can check it by typing git status as below**

****

**To reflect these changes we need to add again in staging area and commit the changes**

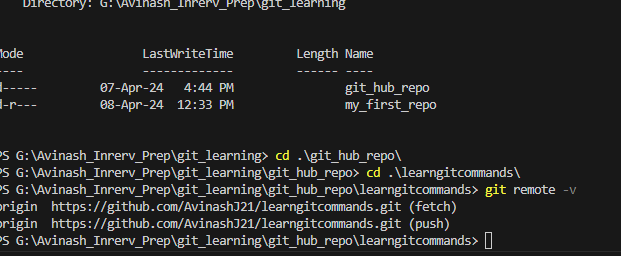
****

**To check log we can also use git log command that retrieves all the actions done to repository**

****

* **Let’s learn how to push local code to a remote branch on GitHub.**
* **We can check for remote branches with the command:**
  + **git remote -v**
* **If you run this command on a cloned repo, you will view the URL of the remote branch, like the GitHub URL.**
* **If there is no connection to a remote branch, then you won’t see a URL.**

**This will work for clone repo from git hub as below**

****

**Now if we want to add file to the repo in git hub then we need to slightly change the command as that of local git**

* **We tell git we want to add a remote branch using the git remote command syntax:**
  + **git remote add name https://url.git**
* **By convention, we call this remote branch the origin branch.**
  + **git remote add origin https://url.git**
* **You then replace the .git URL with the .git URL from the repository you created.**

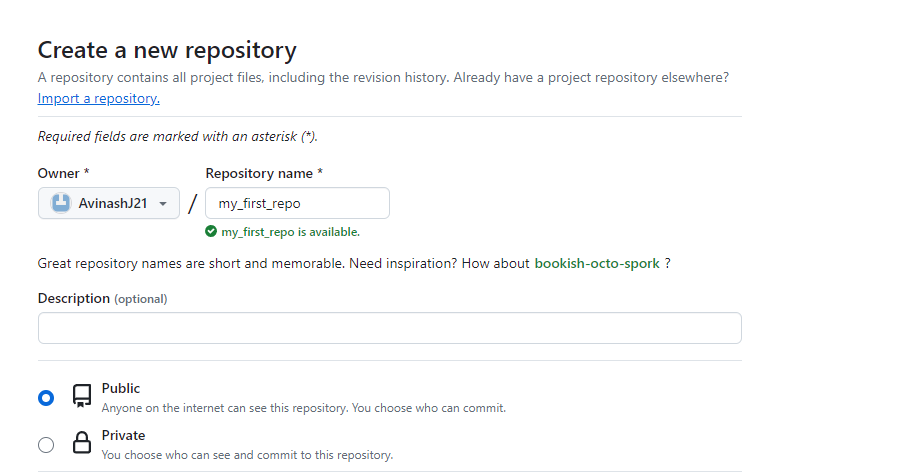
**Extra remote is added in command and url of git hub repo**

* **Important Note:**
  + **We won’t use these commands in the video, but just in case you need them in the future:**
    - **git remote rename <old> <new>**
    - **git remote remove <name>**
* **Once we’ve connected to our remote branch on GitHub, we can push our code to the remote branch.**
* **We tell git to push to the remote main/master branch called origin with the command:**
  + **git push -u origin main/master**

**Git hub has now changed the repo name from master to main**

**Connecting local repo to github**

**Now to add our local repo to GIT hub first create the repo with same name as local repo**

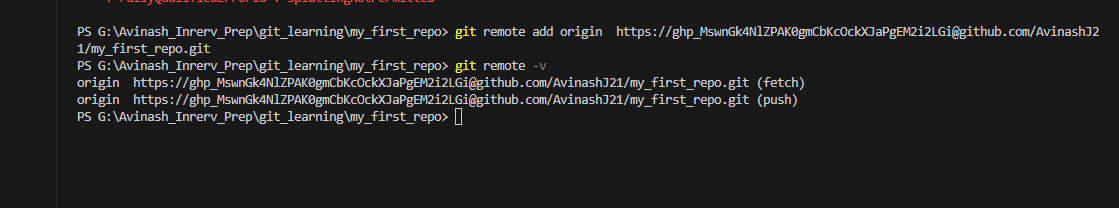
****

**Now to connect local repo to git hub repo**

**git remote add origin** [**https://github.com/AvinashJ21/my\_first\_repo.git**](https://github.com/AvinashJ21/my_first_repo.git)

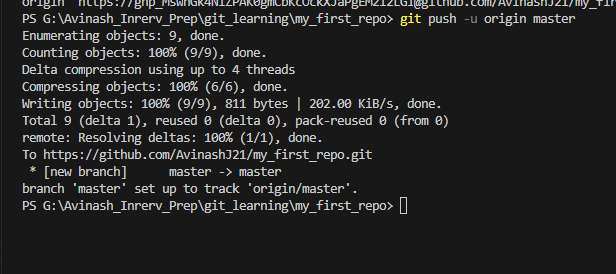
**you may also want to execute above command with access token bcoz for pushing this may keep on asking user authentication**

**git remote add origin** [**https://ghp\_MswnGk4NlZPAK0gmCbKcOckXJaPgEM2i2LGi@github.com/AvinashJ21/my\_first\_repo.git**](https://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@github.com/AvinashJ21/my_first_repo.git)

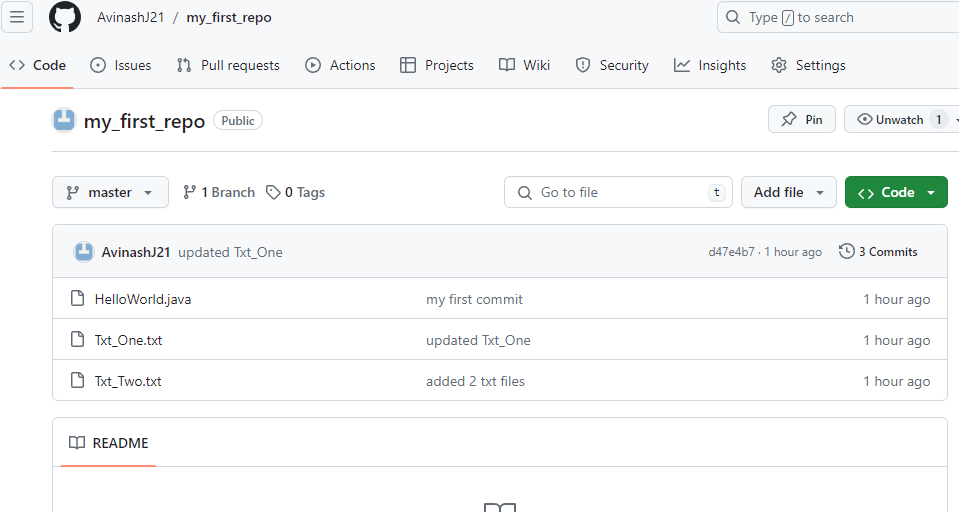
****

**push the local changes to git hub repo we need to use push command**

**git push -u origin master**

****

**So all the local changes are pushed to your git hub repo goto git github refresh your repo**

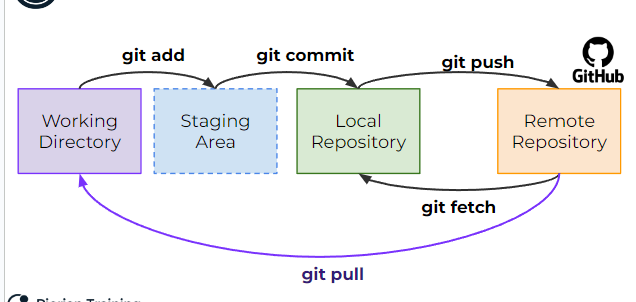
****

* **There are two options of getting repository changes from a remote branch (like the remote branch on GitHub).**
  + **git pull**
  + **git fetch**

**You can get all the remote changes done to repo on your local using fetch/pull**

**Fetch doesn’t make changes in working directory**

**Pull directly make changes to the working directory**

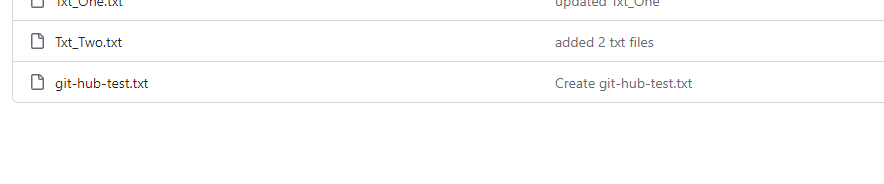
****

* **Using git fetch will download changes from the GitHub remote repository, however you will not see those changes be part of the files you have in the working directory.**
* **Fetch allows you to grab additional work done on the remote master branch, without needing to merge it in your working directory files.**
* **Using git fetch makes sense when you’re working with others and want to see what changes they’ve made but aren’t ready to overwrite your own files yet.**
* **Also if you are simply working by yourself, you may want to just fetch remote changes without overwriting your latest work (later we’ll discover branches are a better way of doing this).**
* **When using fetch, often you’ll just use:**
  + **git fetch**
* **But you can specify to fetch specific branches using:**
  + **git fetch <remote> <branch>**
  + **git fetch origin <branch>**
* **Using git pull makes sense when you want to directly grab changes from the remote branch and directly merge them into your current branch.**
* **This means you will literally update the files in your working directory to match up and merge with the remote branch.**

**We can also check what changes has been done to the remote repo by switching to remote repo**

**Say I’m adding new file to the remote via git hub. I have added new file git-hub-test.txt**

**I**

****

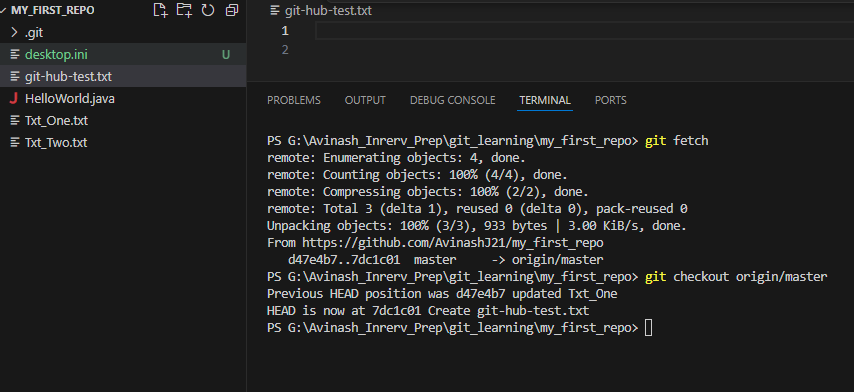
**First we need to run fetch command**

**git fetch**

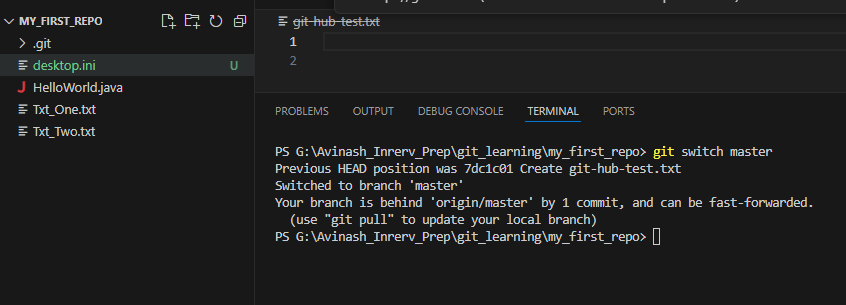
**git checkout origin/master**

**when you run the above commands then new file will start appearing in directory but its still not part of local repo bcoz we have done fetch not pull**

**Its good to know when you are working with many developers what changes has been done to the remote repo**

****

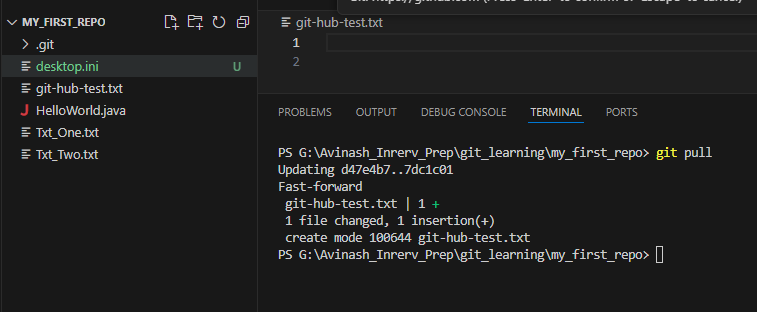
**When you execute git switch master it will switch to local repo changes**

****

**As you can see now that file is removed**

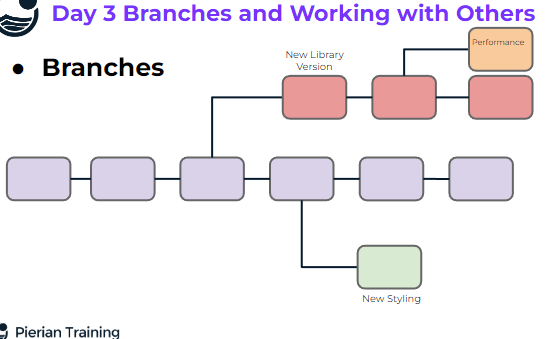
**To get all changes into our working directory then run pull command**

**git pull**

****

**Branches**

* **Branches allow us to organize a repository and split it apart so multiple people can work on it or so a solo developer can work on different aspects of a project on a separated work.**
* **Commit Process**
  + **As we need incorporate the workflows of others or be able to focus on new updates without breaking old code, we need branches.**
* **Branches**
  + **A branch represents an independent line of development.**
  + **Branches serve as an abstraction for the edit/stage/commit process.**
  + **They are a way to request a brand new working directory, staging area, and project history.**

****

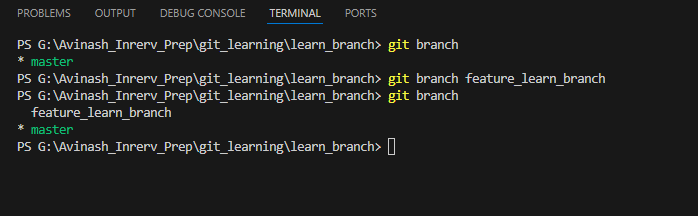
* **Branches**
  + **Upon creating a new repo with git init you create a new branch called the master branch (or main branch).**
  + **This is a branch just like any other, but it’s simply the first one created.**
    - ***Should code pushed to master branch always be in working condition?***
* **Branches**
  + **Master vs. Main**
    - **As we’ve discussed previously, GitHub has changed the nomenclature for this initial branch to be main branch while Git is still using master branch (but this may change in the future).**

**You can also rename any branch (trunk branch).**

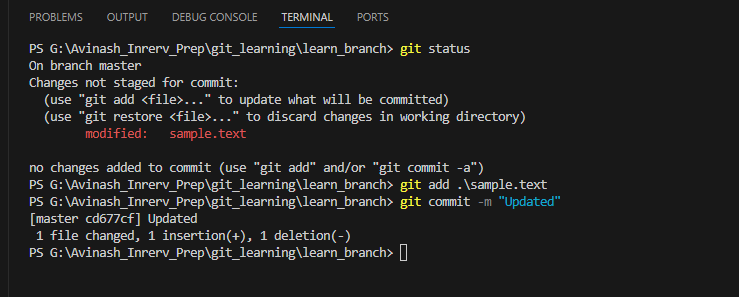
* **Branches**
  + **As we work more with branches, you will probably notice a term show up during your commits: HEAD.**
  + **When viewing the most recent commit using git log you may see:**
    - **commit 05as..3e2 (HEAD -> master)**
* **Git Branch Commands**
  + **Create a New Repo**
  + **Add File**
  + **Create a New Branch**
    - **git branch <branch\_name>**
  + **Report Branches**
    - **git branch**
  + **Switch Branches**
    - **git switch**

**lets create a new branch**

**to do that first will create local repo add files then commit**

****

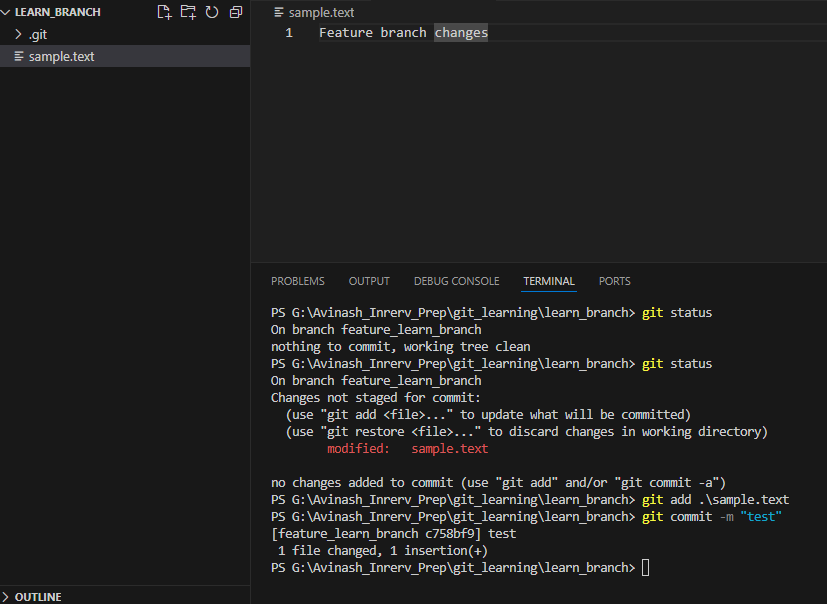
**I updated the sample.txt note we are still in the master / main branch**

****

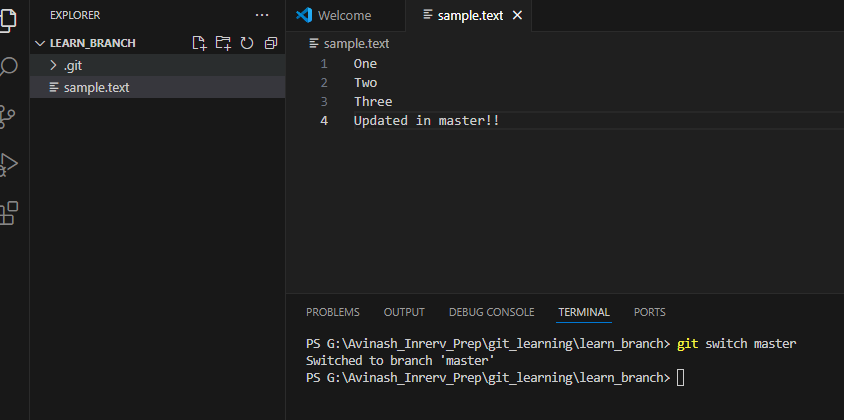
**Now lets switch to our branch**

**git switch feature\_learn\_branch**

**lets make some changes in the sample.txt file**

****

**Now lets switch to master branch notice it shows the changes related only to master branch since we haven’t merged our changes in master**

****

* **Renaming a Branch**
  + **git switch branch\_to\_rename**
  + **git branch -m new\_name**
    - **You must be checked out on the branch you will rename.**

**git checkout -b <branch\_name> to checkout the branch created on github to local**

**git pull origin master: to take/update all the changes from github master repo**

**git pull origin <branch\_name>**

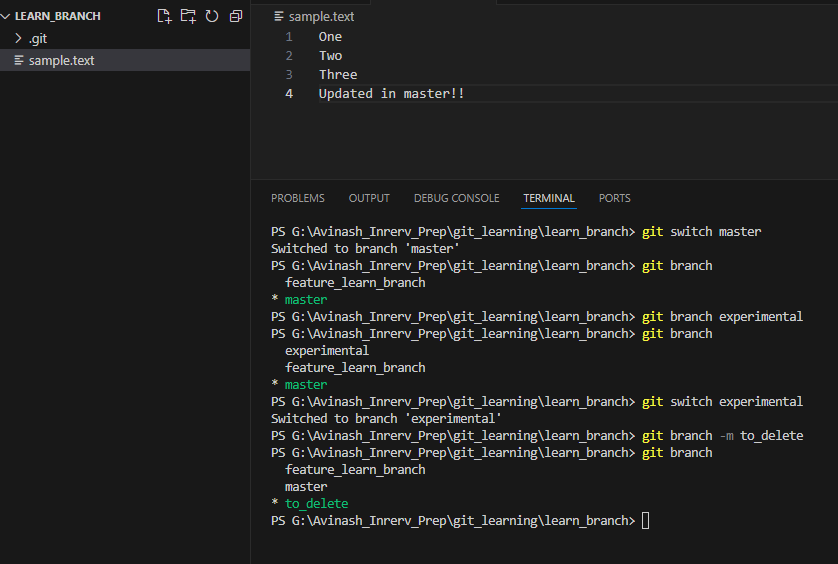
**git push origin <branch\_name> sometimes it ask for credentials**

**git merge master <branch\_name>**

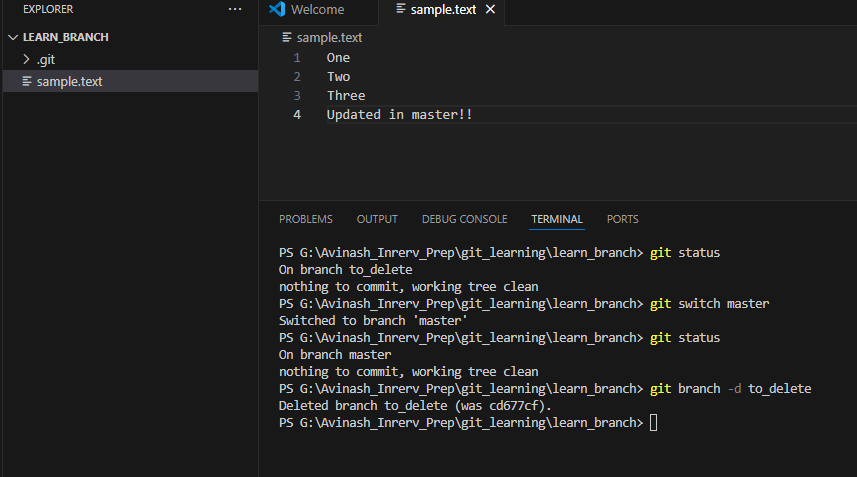
**Then to merge this update into another branch use merge**

* **Deleting a Branch**
  + **git branch -d branch\_to\_delete\_name**
    - **You can not delete a branch you are checked out at.**
    - **You also will get a warning if the branch is not merged.**
      * **You can confirm you want to do this anyways with -D**

**To rename and delete learning lets create another branch experimental then rename it as below**

****

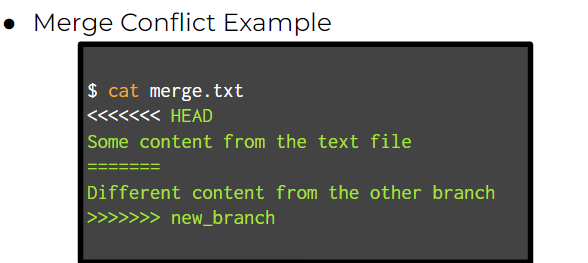
**To delete branch we cant be in same branch so switch to master or another branch**

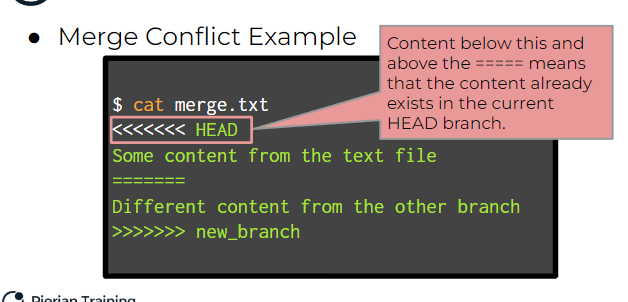
****

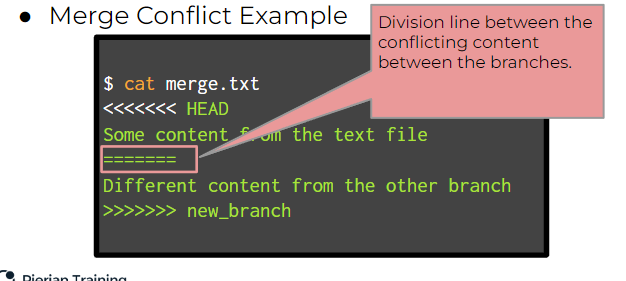
**Merging**

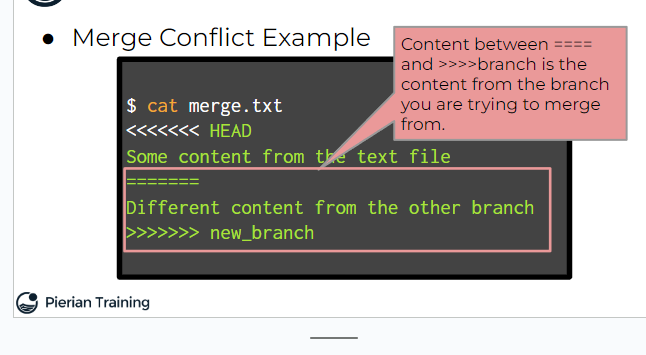
**So there are some changes in master branch and we want merge these changes in another branch so that all are on same line of development**

* **However, there will be many instances where there are conflicts, for example changes in the file on lines that are different between the branches.**
* **These are known as merge conflicts, and we need to resolve (fix) the conflicts between the branches in order to merge them.**
* **Git will warn you about files in conflict.**
* **Then you must edit the files in order to remove the conflicts.**
  + **Fortunately, Git also provides specialized markdown to indicate the differences between the files and what differences come from which branch.**
  + **Modern editors (e.g. VS Code) have syntax highlighting to reflect this.**

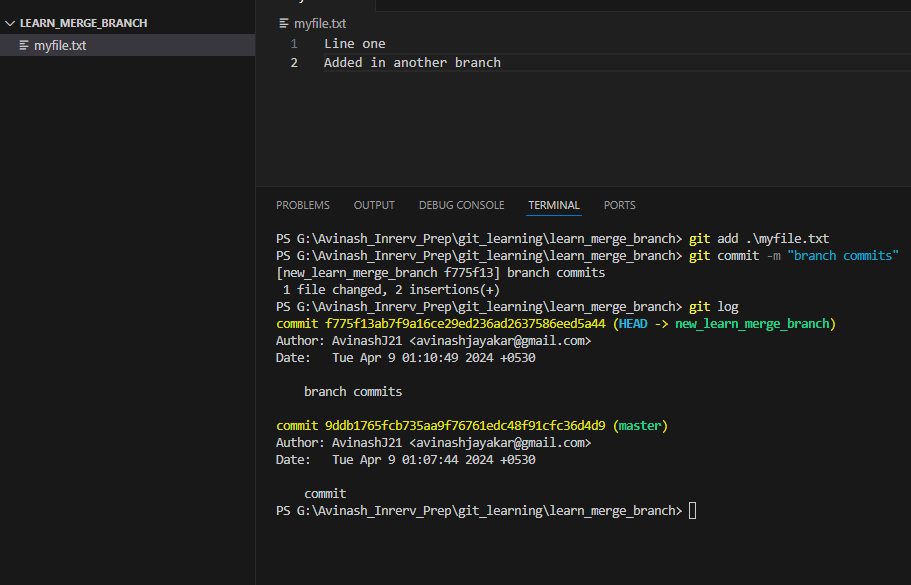
****

****

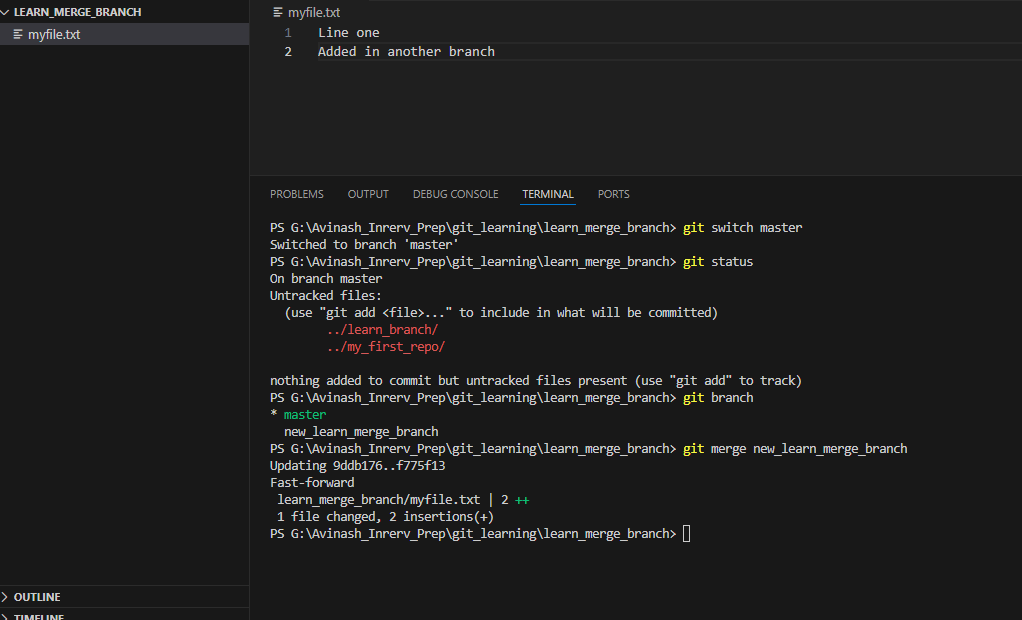
****

****

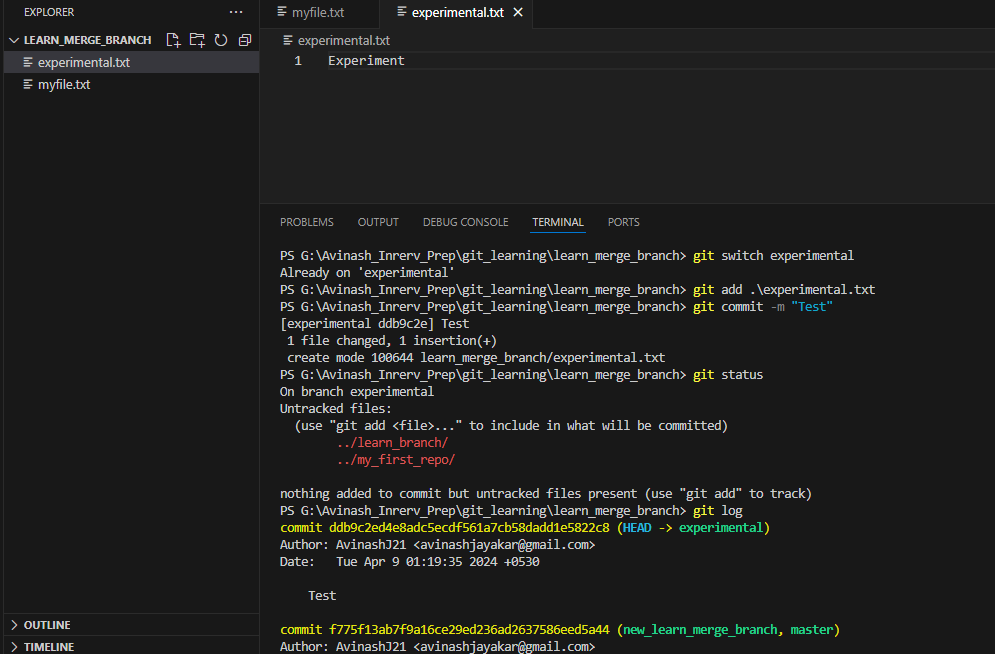
**Lets create new repository and add a file myfile.txt**

****

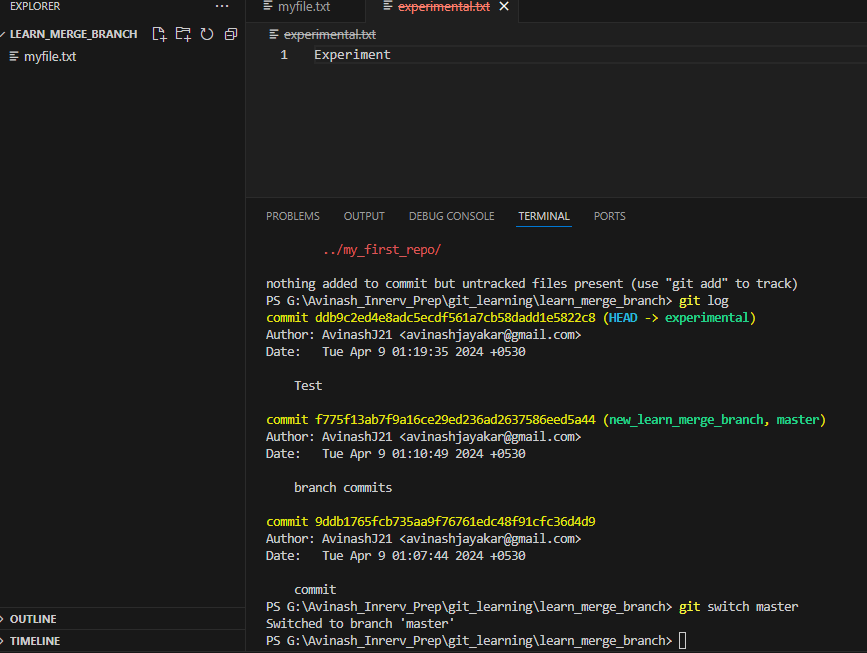
**Lets merge branch related changes to master. I need to switch to master**

****

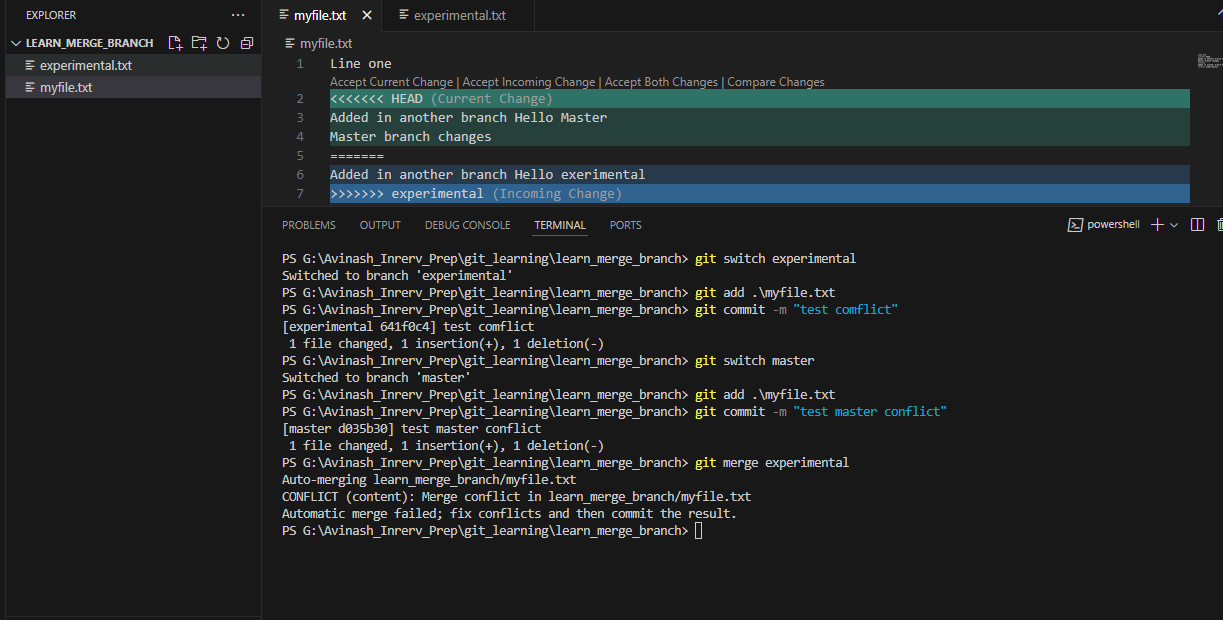
**Create another branch experimental and add new file**

****

**Now lets this branch changes in master**

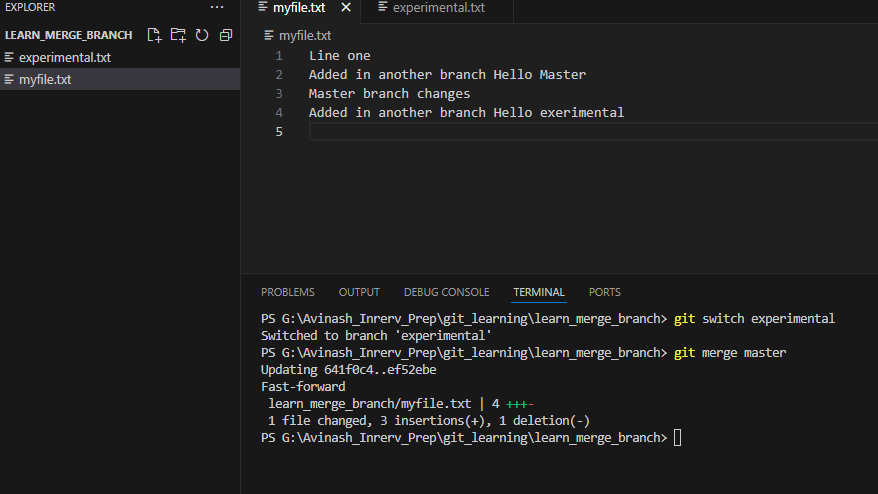
****

**Now lets make chages in same file at same line notice you are getting conflicts**

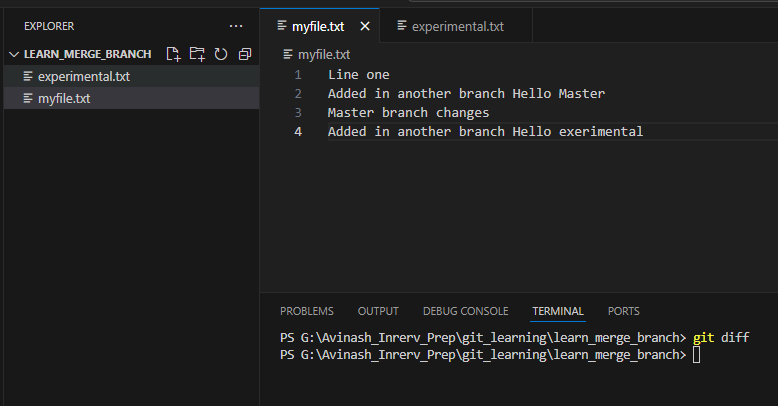
****

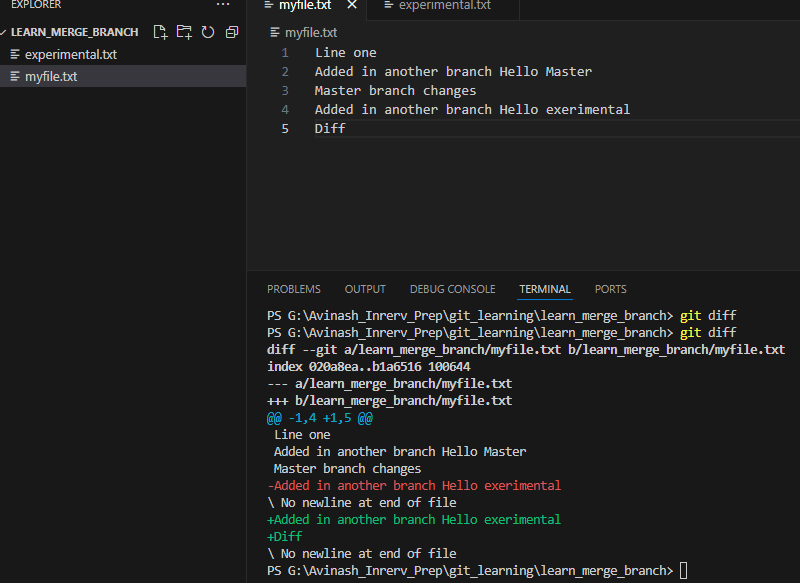
**VS code will provide with bunch of options to resolve conflict you can keep both changes , accept current change, accept incoming change accept both changes, compare changes**

**I have accepted both the changes hence changes from other branch will be added to new line so conflict will be resolved and now try to switch to experimental branch and merge master you will see both changes as below**

****

**git diff : Used to show differences from the previous commit till changes done in current branch.**

****

****

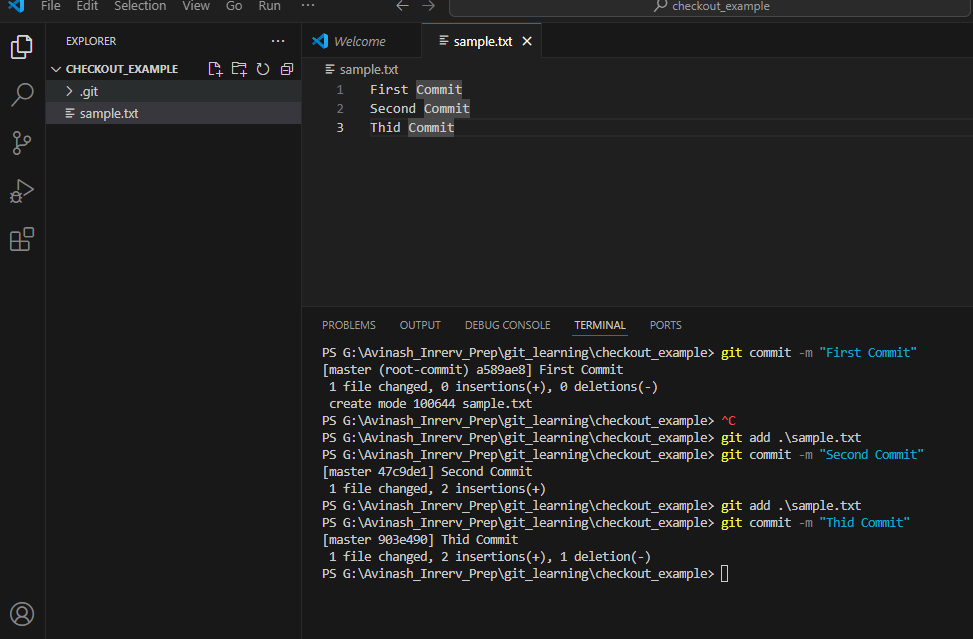
* **We’ll discuss:**
  + **git checkout and Detached HEAD**
  + **git restore**
  + **git reset**
  + **git revert**
* **Keep in mind that you don’t use these commands as often as the other commands we’ve learned so far, but they are still important actions to know!**
* **This is actually a very versatile command, so versatile in fact, that developers complained it was used for too many different actions, thus new git commands were created, such as git switch.**
* **git checkout**
  + **A "checkout" is the act of switching between different versions of a target entity.**

**The git checkout command can operate on three distinct entities: files, commits, and branches.**

* **git checkout**
  + **We can check out a particular commit by specifying its hash, we can get hashes from the git log command and we can also see the abbreviated hash using:**
    - **git log --oneline**
  + **Then we can provide the has as:**
    - **git checkout #######**

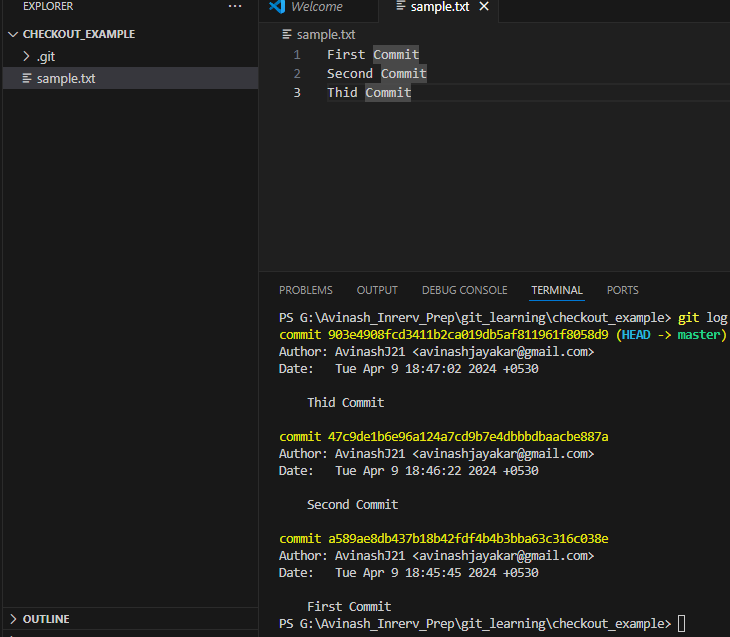
**git checkout is used to travel back in time to view the particular commit and changes to that state**

**As below I have committed the changes for 3 times**

****

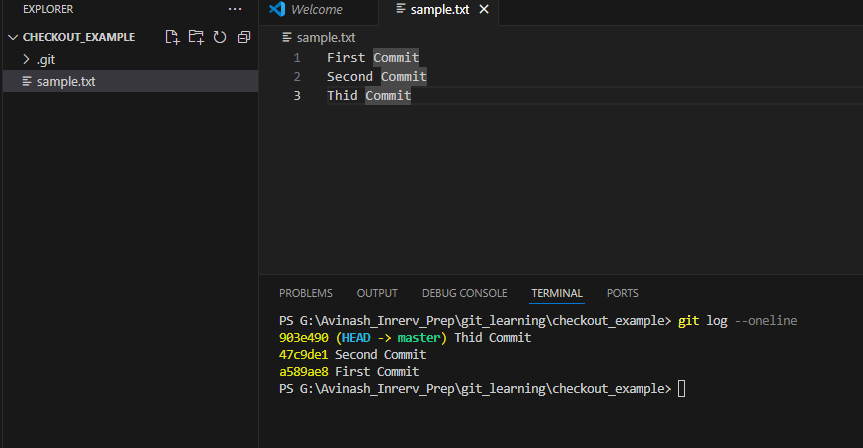
**Now I can view the state from second commit using checkout. Lets see the log**

**git log**

****

**We can also see the short form of these commit known as hash**

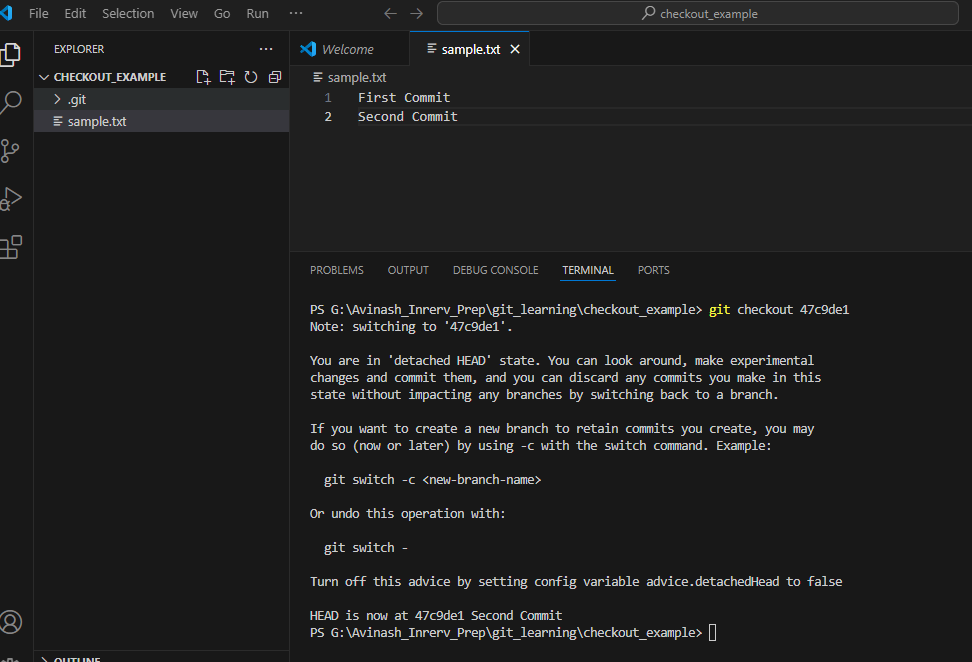
**git log –oneline**

****

**Lets checkout to second commit**

**git checkout 47c9de1**

**After checking out Head (pointer) will be in detached state**

****

**Now if I want to move again to current changes use below command**

**git checkout master**

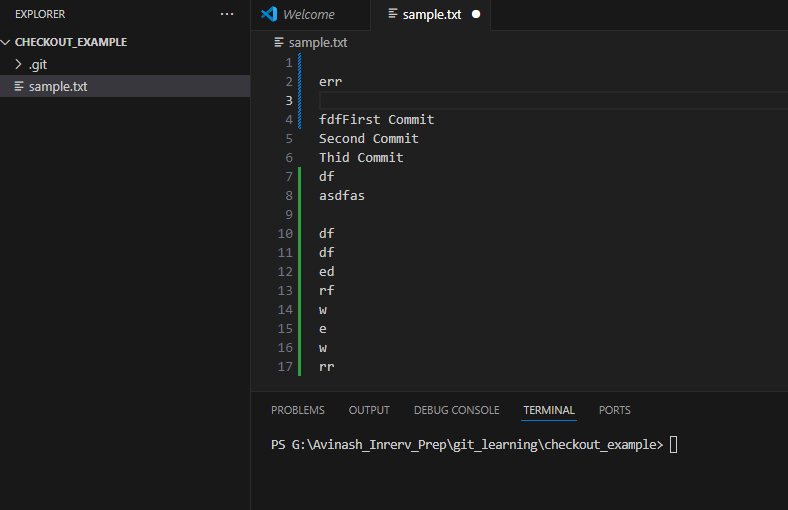
**Git Restore**

* **Imagine you’ve been working on a file that is already part of a commit.**
* **You know the code was working at the point of the last commit, however now the code is totally broken, and you just can’t seem to fix it.**
* **You’ve written too much code to just run Ctrl+Z! How to fix this?**
* **Git restore**
  + **We can restore a file to its state at the previous most recent commit using the git restore command:**
    - **git restore file\_name**
  + ***Warning:***
    - **You can not undo a git restore command, since your changes were not committed!**
* **Git restore**
  + **We can restore a file to its state at the previous most recent commit using the git restore command:**
    - **git restore file\_name**
  + ***Warning:***
    - **Think of this command as an ultimate “Ctrl+Z” restoring files to their previous commit.**
* **Technically speaking git restore will restore the file back to the HEAD, which typically we have pointing to the most recent commit in the branch.**
* **Git restore**
  + **This actually gives us even more flexibility in our restore procedure, we can restore a file to any commit in the log.**
  + **We state the number of commits from the HEAD to go back to:**
    - **git restore --source HEAD~N file.txt**
* **Git restore**
  + **Finally, git restore also allows us to unstage files that we had already added to the staging area using git add.**
  + **We can do this with:**
    - **git restore --staged filename**

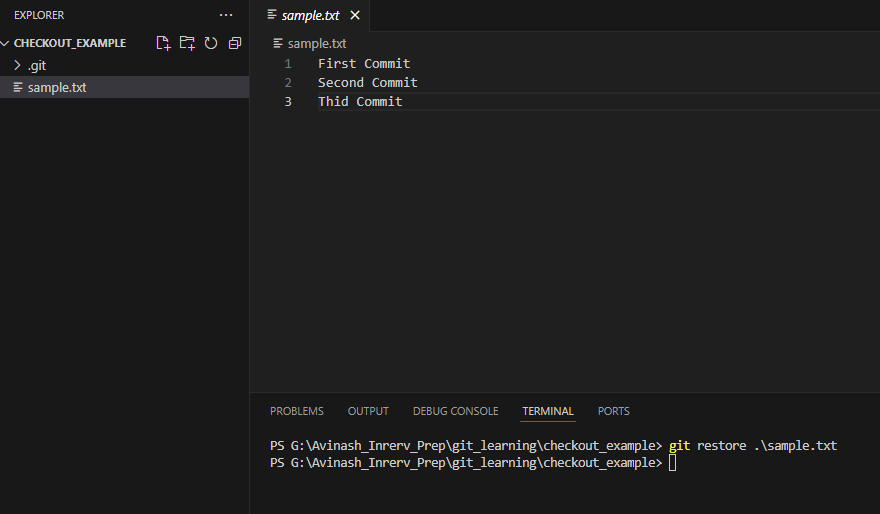
**Working examples**

**Sometimes in your code you have made lot of changes and also it becomes impossible to undo changes by ctrl+z so in that case we can make use of git restore cmd**

**git restore <file-name>**

****

**Close the file then restore command and then open txt**

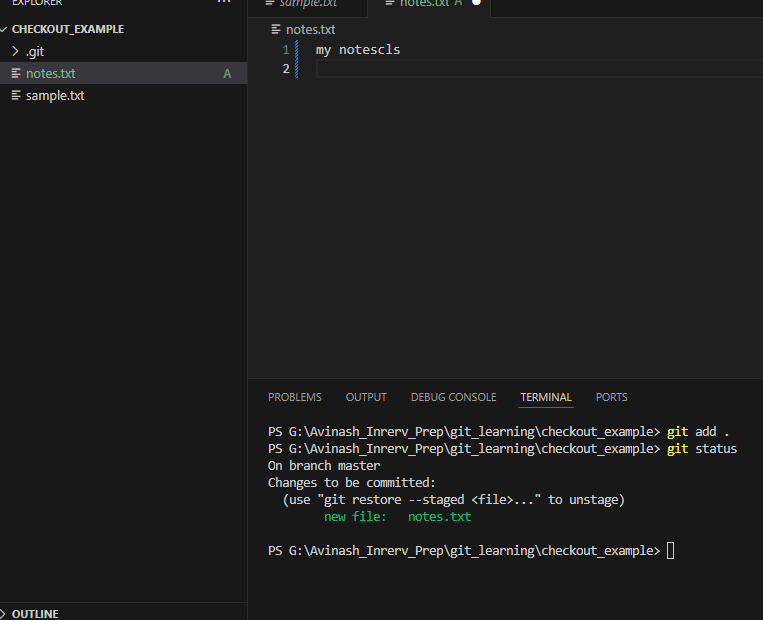
****

**You can also restore by using number commits as well say previously you have committed the file and there are 3 counts you can restore file by previous 2 count commit**

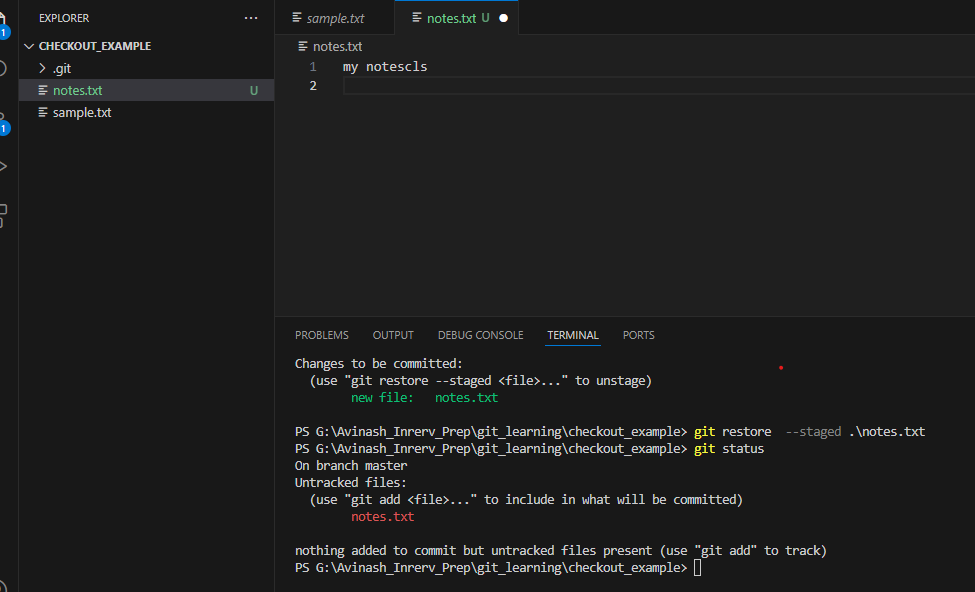
* + - **git restore --source HEAD~N file.txt**

**N stands for count**

**We can also restore from the files from satged area say for example you have added the files by mistake in staging area as below**

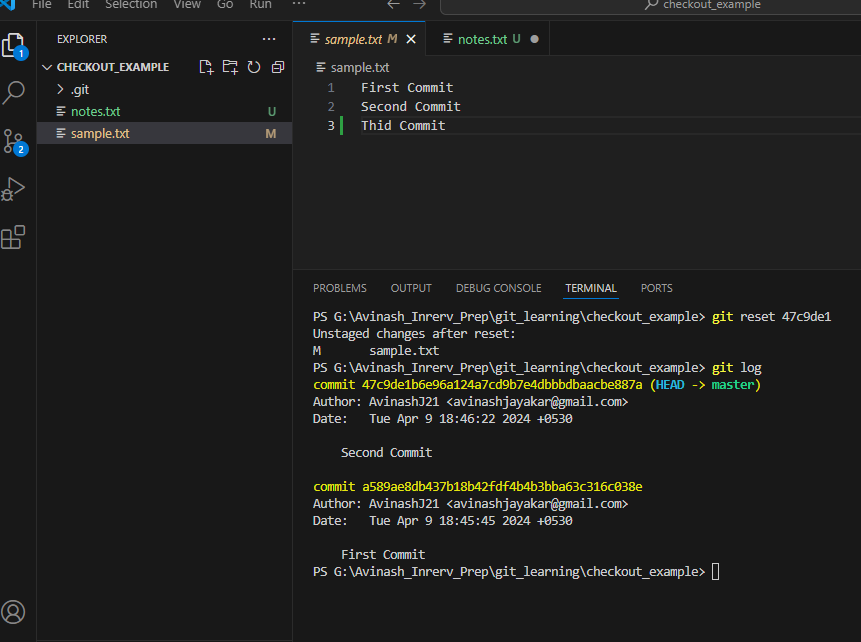
****

**As you can see mistakenly we added notes.txt**

****

* **Git reset allows us to remove commits and “reset” the branch.**
* **There are two main types of git reset calls:**
  + **git reset #######**
    - **Removes commits in front of the specific hash called, files unchanged.**
  + **git reset ####### --hard**
    - **Removes commits *and* the changes in the files.**

**This is actually very helpful when you accidentally committed in wrong branch and you want to undo the commit but also keep all file changes**

****

**As you can see it reset to second commit but changes related to third commit is still there**

**If you also don’t want to keep file related changes then append –hard**

**Git reset <commit hash> --hard**

* **Let’s explore the last command for undoing changes, git revert.**

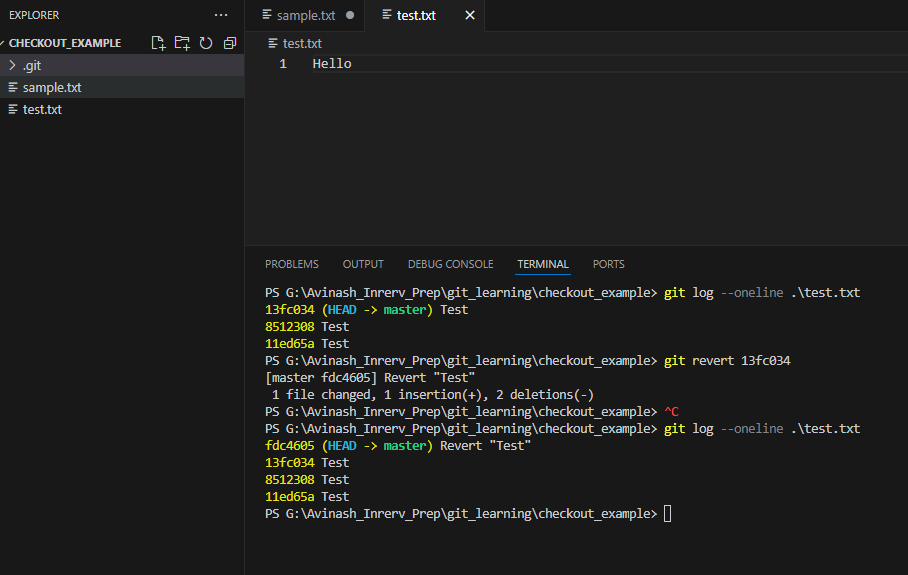
**The git revert command will create a new commit that undoes work from previous commits, but keeps those commits in the branch**

**Git revert doesn’t change the project history, which makes it a “safe” operation for commits that have already been published to a shared repository.**

* **Git revert is a safer alternative to git reset in regards to losing work.**

**Bcoz git reset looses all previous commits and hence merging becomes harder**

**After revert git will automatically create commit as you can see fdc## is created**

****