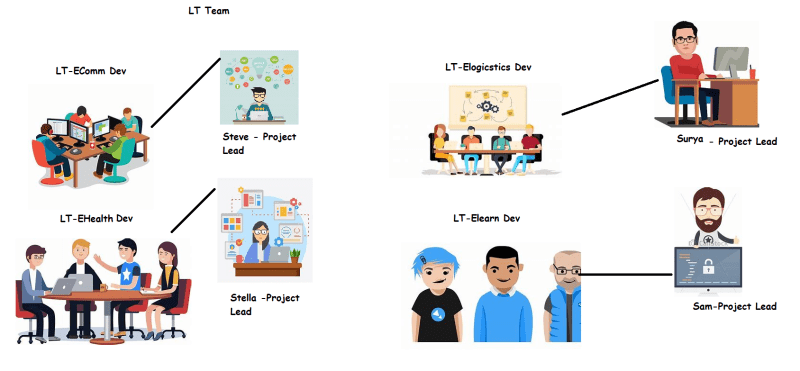
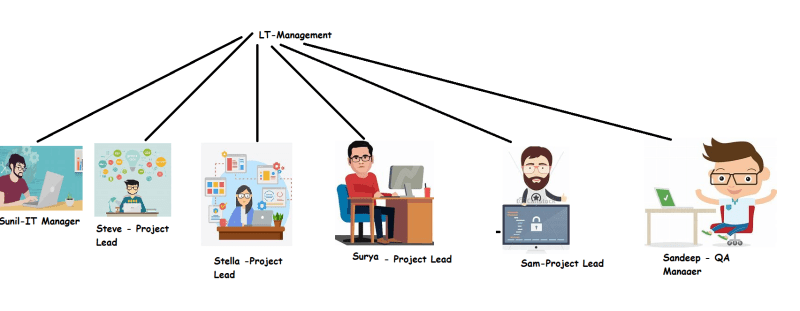
DevOps Class *CI****/****CD* (GIT) – 02/Feb/2021

Scenario of Application Development and Deployment in Learning Thoughts

* LT (a ficticious organization) is working on 4 projects
  + LT-ecomm:
    - This is ecommerce project developed in Java Tecnologies
  + LT-ehealth:
    - This is a Healthcare project developed in .net Technologies
  + LT-elogistics:
    - This is a logistics project developed in python Technologies
  + LT-elearn:
    - This a elearning project developed in node js
* The developement and project lead teams are 
* LT-Management Ladder and QA and IT Team 
* Plan for deploying Applications into various environments
  + All the applications developed in LT will have to undergo QA in the following environments
    - Unit Testing/Developer
    - Functional Testing / System Test Env
    - Performance Testing/ Perf Test Env
    - UAT/Pre-Prod
    - Manual Testing Environment
  + Then the application can be deployed to Production
  + All the testing should be automated- Sandeep and his team is responsible for that (apart for Manual Testing Environment).
  + Unit Testing should happen on the developer system and for every change submitted by developers
  + All of the changes done by developers during one day are consolidated and a System Test Env has to be created and Automated Functional Tests should be executed during night and results should be send to all the team members
  + Manual Testing Environment is updated along with Functional Testing environment every night
  + On Every Friday night the Performance Testing Environment should be created and all the performance tests should run over weekend and on monday performance test results should be published.
  + Once in two weeks all the work done by developers in that two weeks will be consolidated and a small release will be made and UAT environment is created where UAT Testing happens
  + Once the UAT is succesful, these changes will be published to Production environment (Prod is updated once in two weeks)
* Summary: There are actions to be done
  + Whenever developer submits code
  + On some schedules (Every weekday at 9:00 PM, Every Friday at 10 PM, Every alternative Thursday, Every alternative Monday)

#### How to realize this

* We need create this pipeline
  + to create test environments
  + to execute tests
* We need to understand the system where developers are submitting the code (Version Control System)
* To Create pipelines to automatically create environments and execute the tests we need some system => CI/CD Engine (Jenkins, Azure DevOps (vsts))

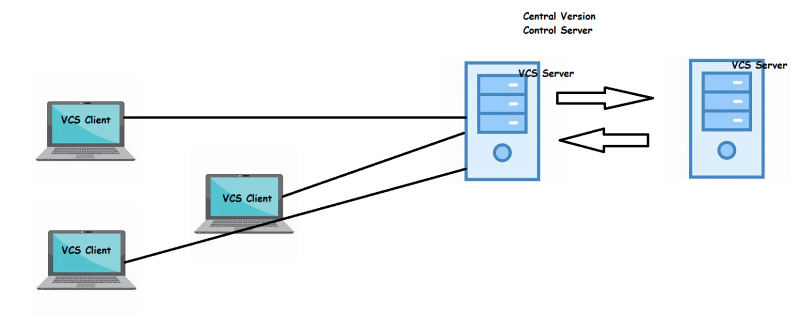
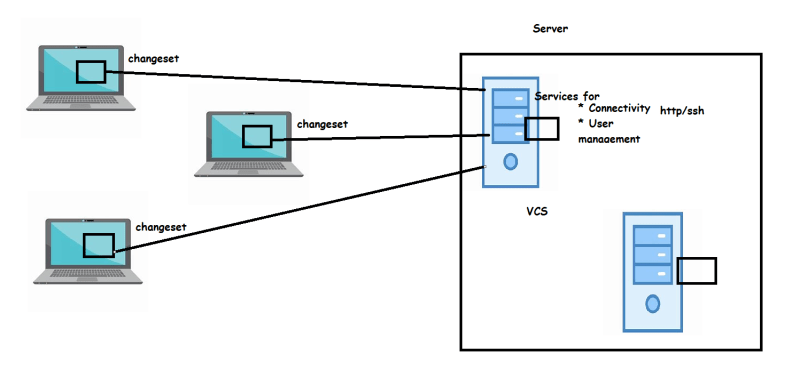
### Next Steps:

* Understand Version Control Systems and Learning a VCS called as GIT

### Exercises:

* Create any Cloud Account (Free tier)
  + AWS [Refer Here](https://www.youtube.com/watch?v=z95MhW1gAcA&list=PLuVH8Jaq3mLszrC7lv68a0VcrDripW-HK&index=1)
  + Azure [Refer Here](https://www.youtube.com/watch?v=MdDOc9OPVDA&list=PLuVH8Jaq3mLuqXuGs6aeqxhuvCYSzB1kT&index=1)
* Softwares:
  + [Refer Here](https://www.youtube.com/watch?v=mRILfUNbsIo&list=PLuVH8Jaq3mLud3sVDvJ-gJ__0zd15wGDd&index=14) for visual studio code and git
  + [Refer Here](https://www.youtube.com/watch?v=qLVn2EvPsYc&list=PLuVH8Jaq3mLud3sVDvJ-gJ__0zd15wGDd&index=11) for Windows Terminal
* Listen to 1-7 videos of [Refer Here](https://www.youtube.com/watch?v=dcWAf6FOOc0&list=PLuVH8Jaq3mLud3sVDvJ-gJ__0zd15wGDd&index=1)

How is Source Code Managed? – 03/02/2021

* Multiple developers work on the same source code for the project
* To make this possibility a reality, we can use
  + Shared Folder:
    - This is a simple approach
    - Problems:
      * Two People cannot work on same file at same time
* To make source code management effective, users should be able to
  + go back in history to find out changes
  + We need versions/Revisions for every change by developer
  + History of all the changes
* Version Control System also known as Source Control is a software that
  + Manages changes to software code by maintaining versions for every change
  + Allows multiple users to use the Source Code parallely and submit their changes
  + Backup and Restoration options in the case of server failures.
* Generations of Version Control Systems
  + First:
    - Networking: None
    - Operations: One file at a time
    - Concurrency: Locks
    - Examples: SCCS, RCS
  + Second: Centralized Version Control Systems
    - Networking: Centralized
    - Operations: Multi-files
    - Concurrency: Merge befor Commit
    - Examples: CVS, Source Safe, Subversion, Clear Cases, Team Foundation Server 
  + Third: Distributed Version Control Systems
    - Networking: Distributed
    - Operations: Changesets
    - Concurrency: Commit before merge
    - Examples: Git, Mercurial 

GIT

* Linus Torvalds creator of Linux developed own tool based on the lessons learned from past experience and had some design goals to build a distributed version control system
  + Speed
  + Simple
  + Fully distributed
  + Ability to handle large projects like linux kernel effeciently
  + Support for non-linear development
* Installing Git
  + Windows
    - Install Chocolatey [Refer Here](https://chocolatey.org/install)
  + Linux:
    - Ubuntu: sudo apt update && sudo apt install git -y
    - RHEL: sudo yum install git -y
  + MAC:
    - When you install xcode we get git [Refer Here](https://apps.apple.com/us/app/xcode/id497799835?mt=12)
    - Install homebrew [Refer Here](https://docs.brew.sh/Installation)

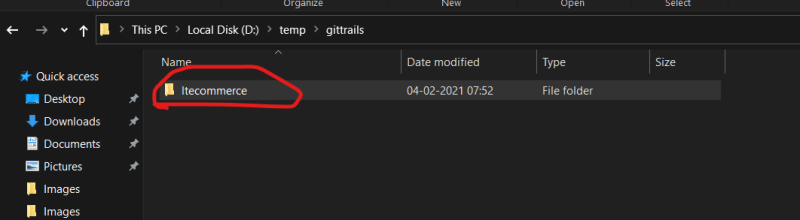
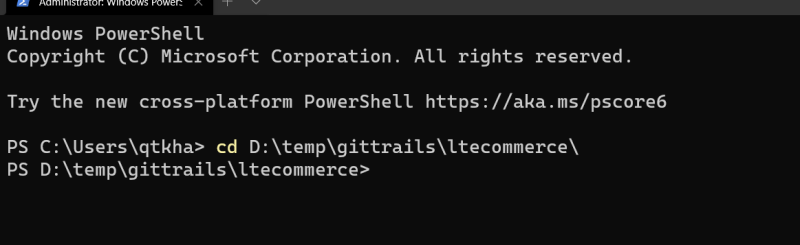
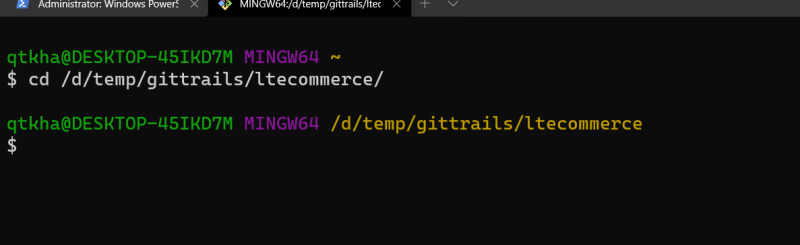
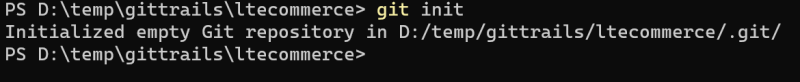
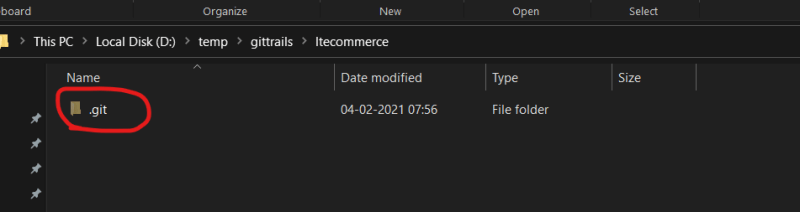
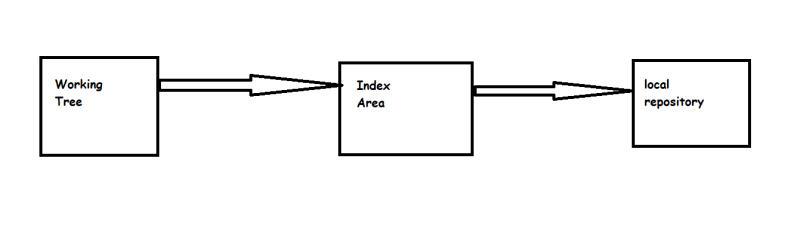
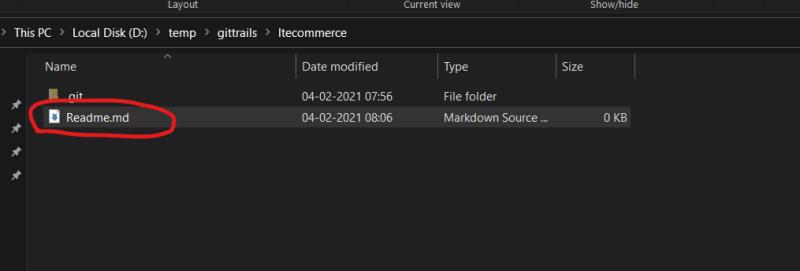
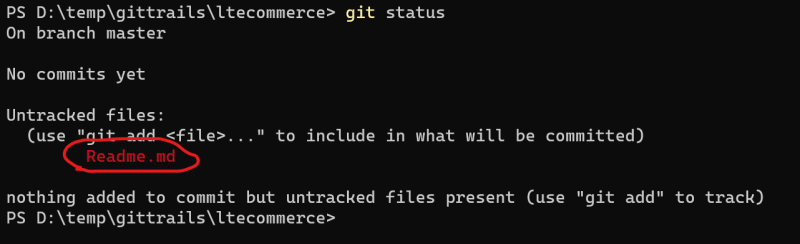
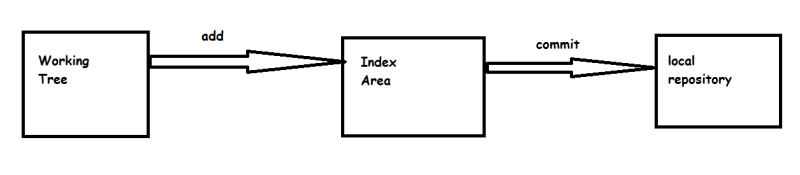
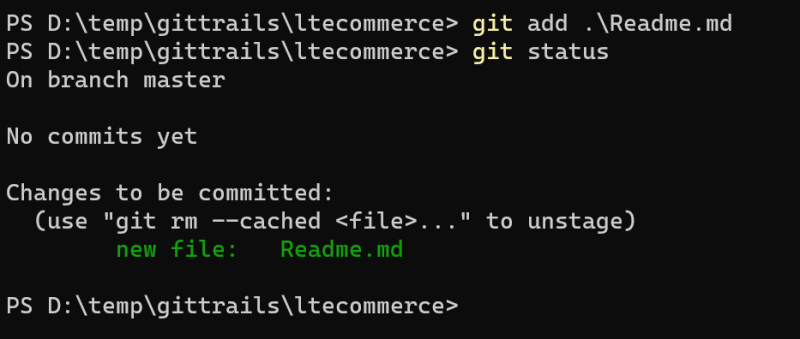
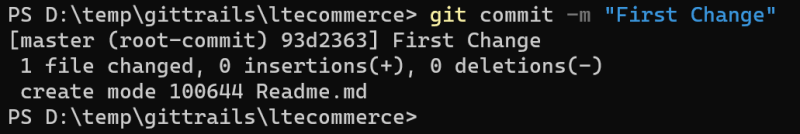
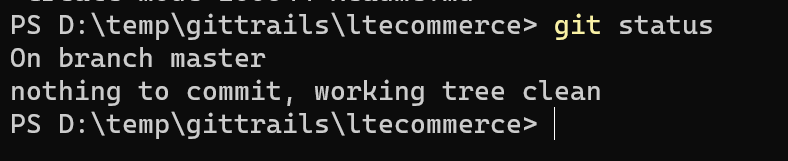
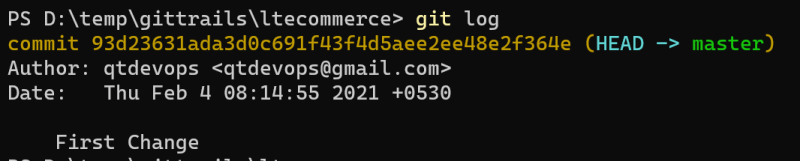
brew install git

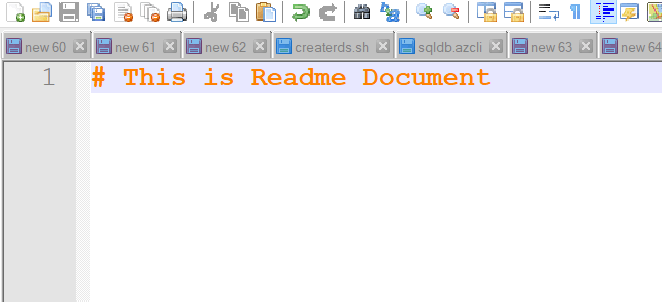
## Git Contd.. 04/02/2021

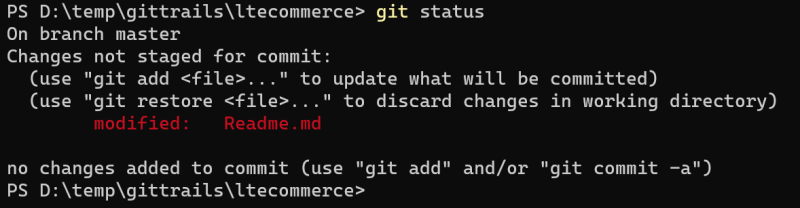
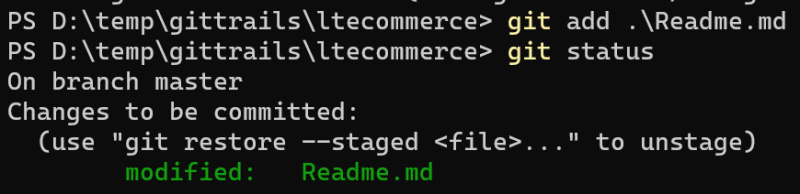
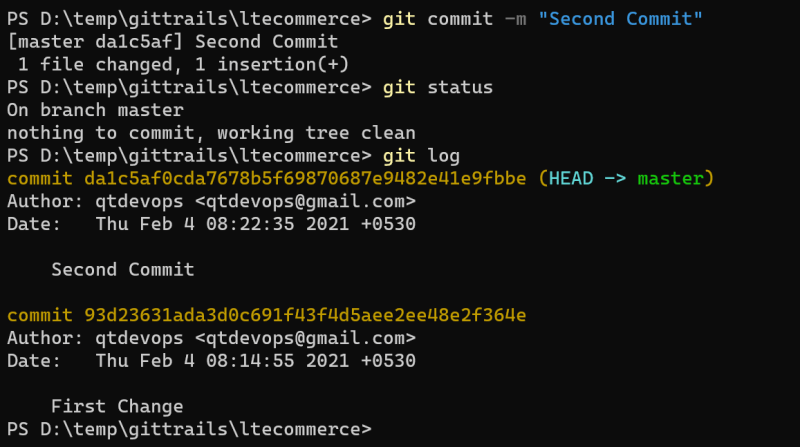
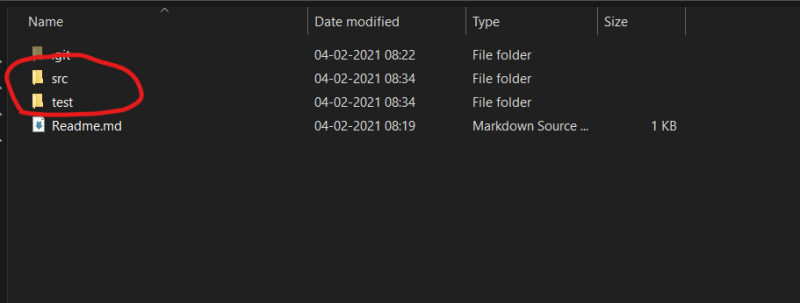
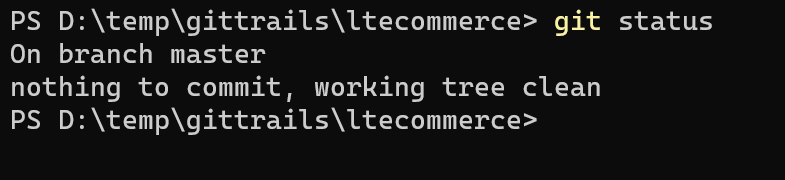
### Terms

* Repository: In short repository is totality of the history of your software project.

Practical Scenarios

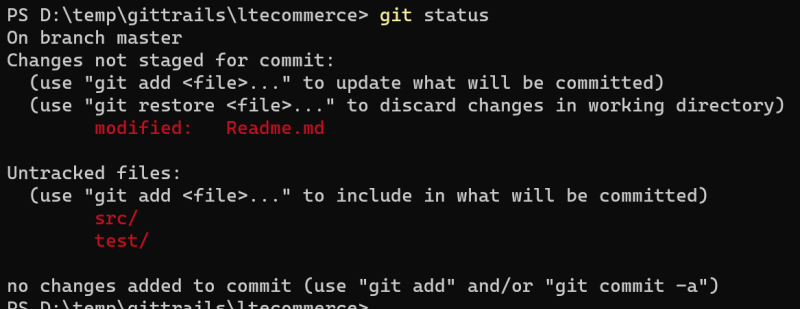
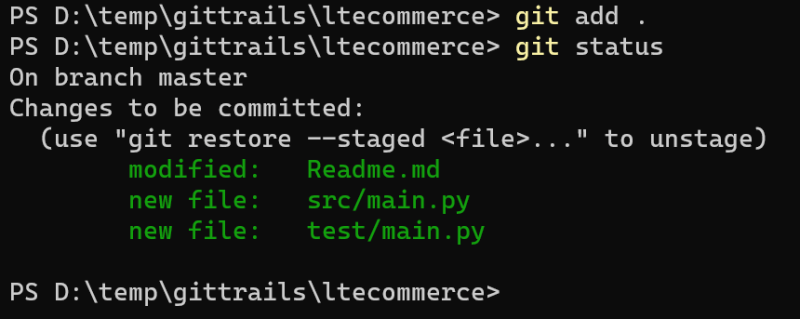
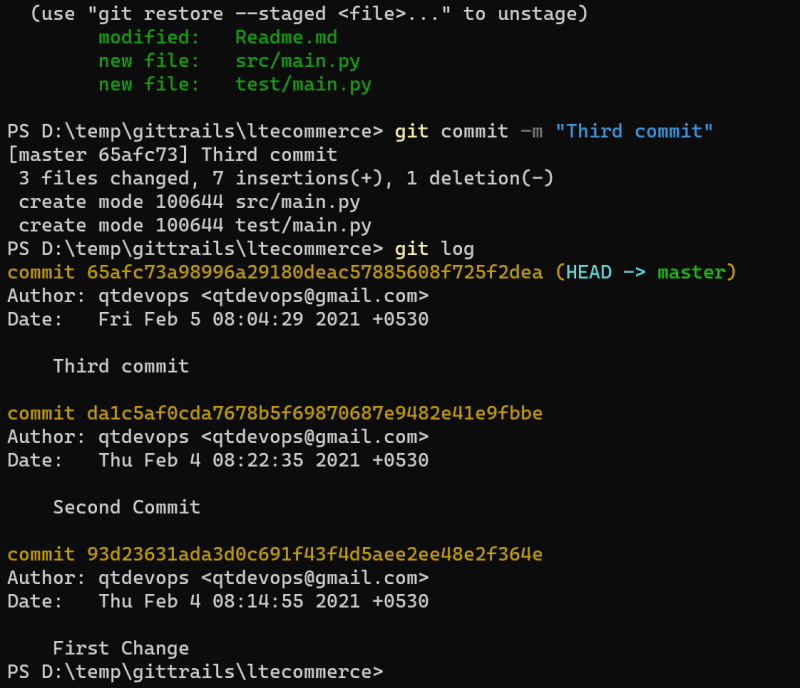
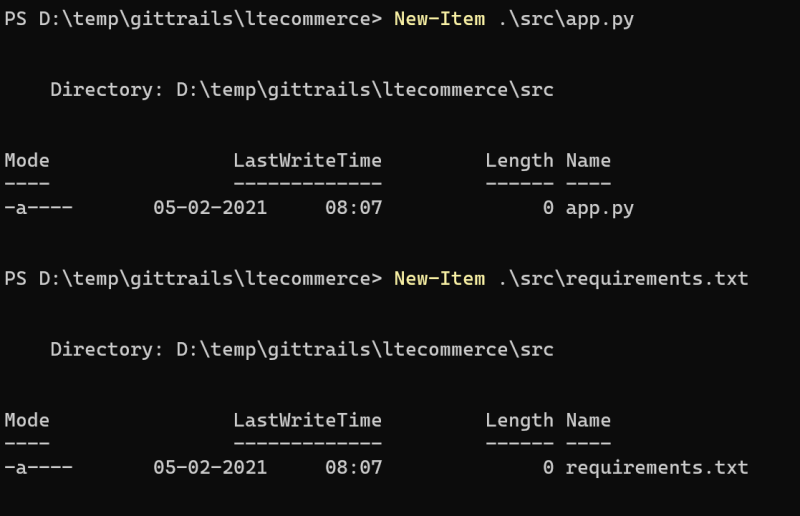
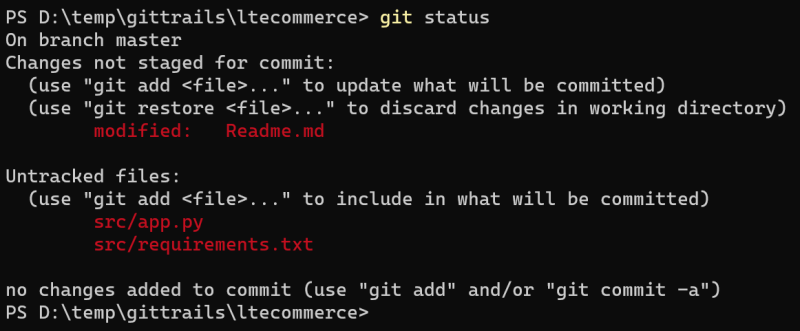
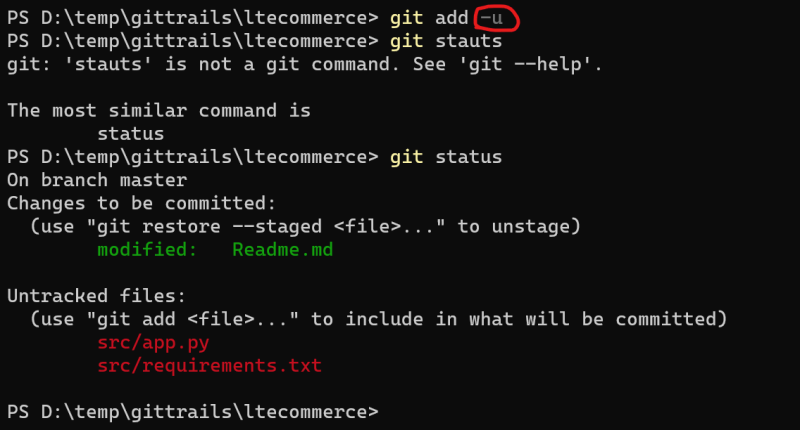
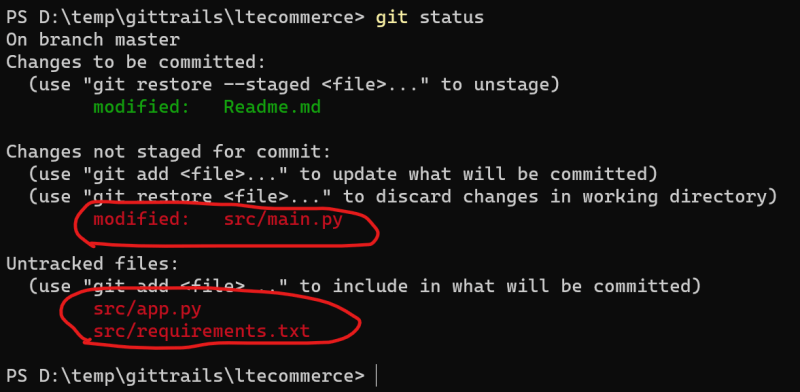
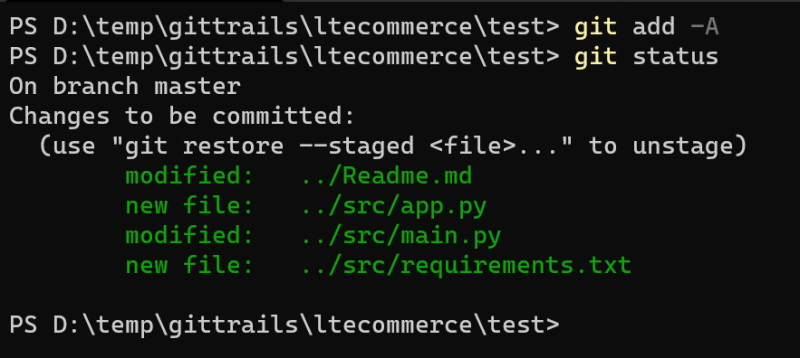
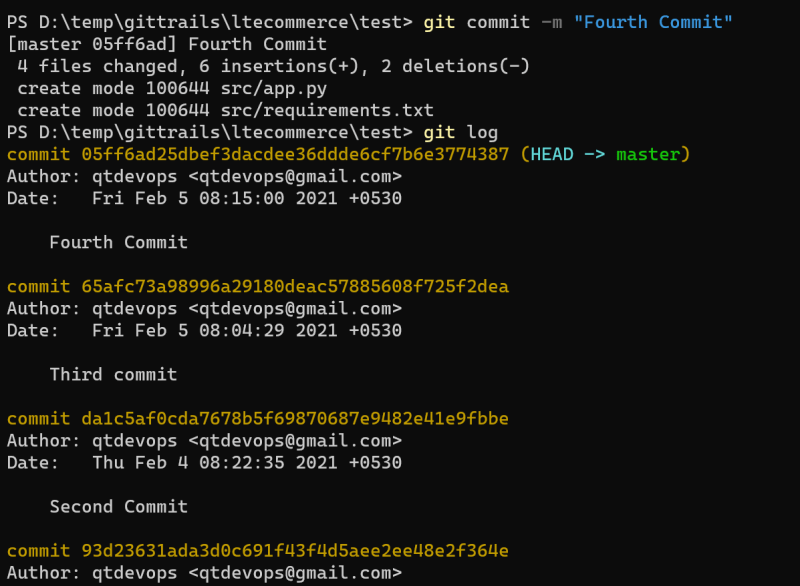
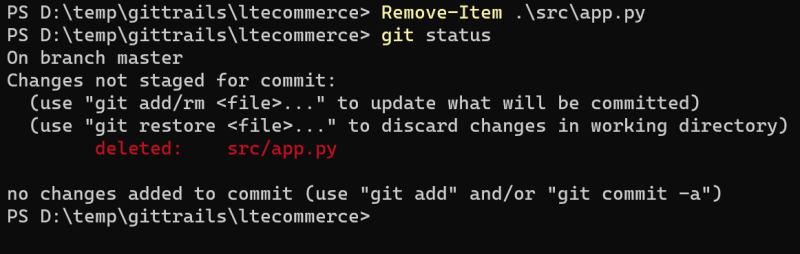
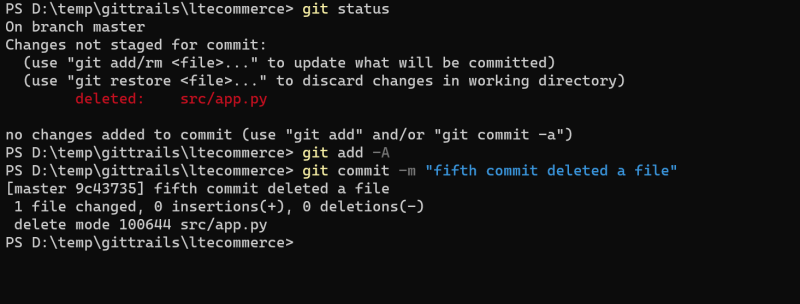
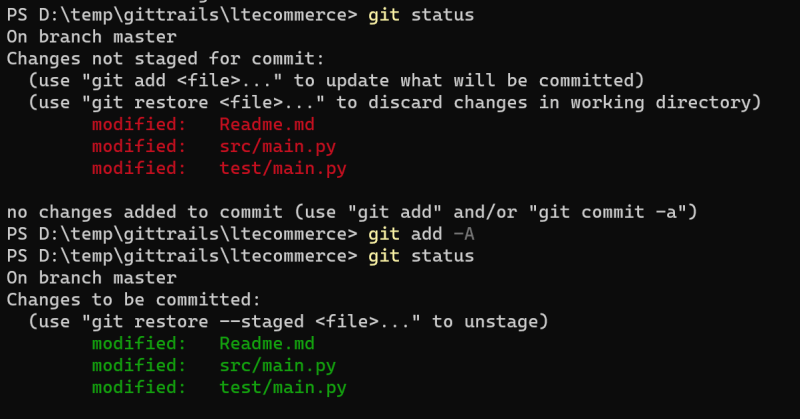
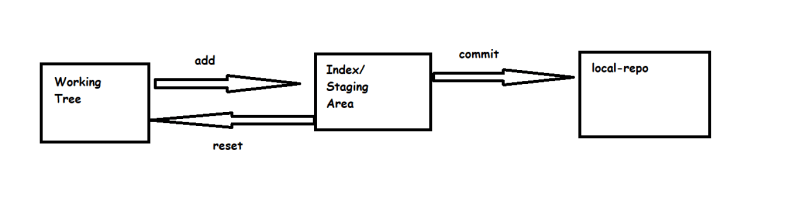
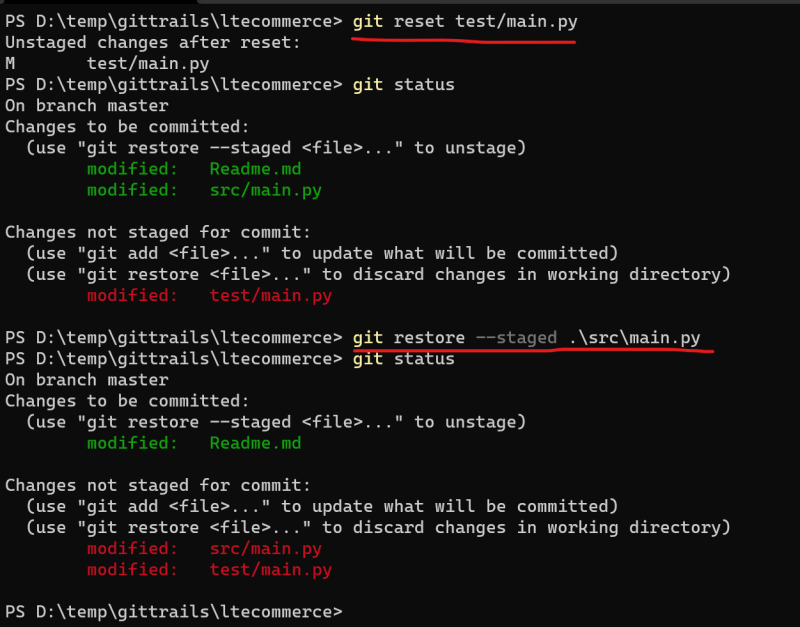
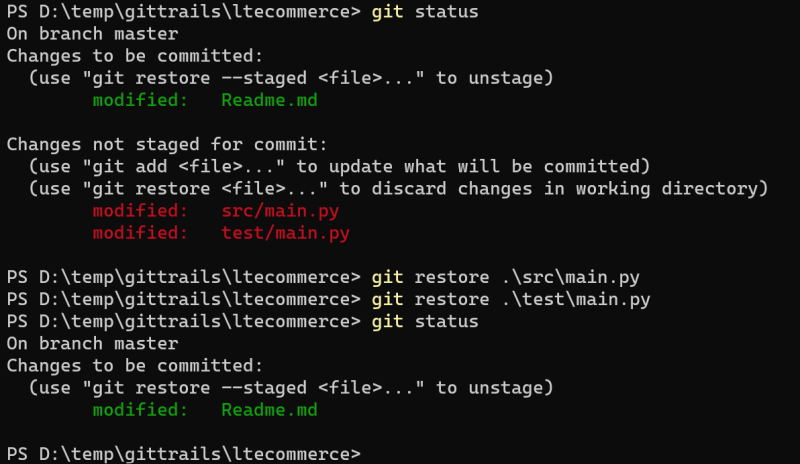
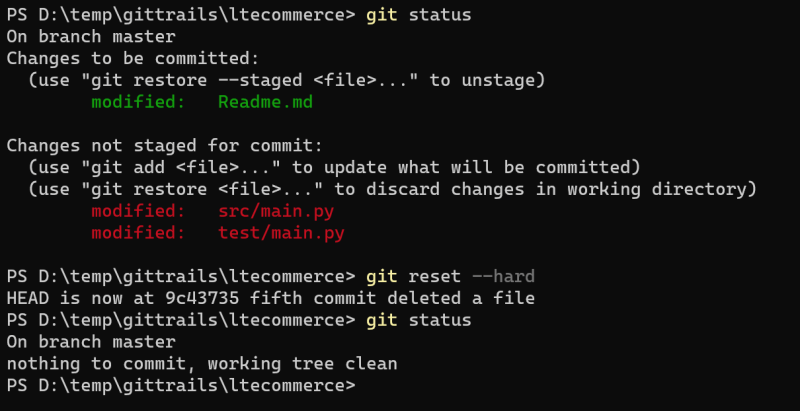
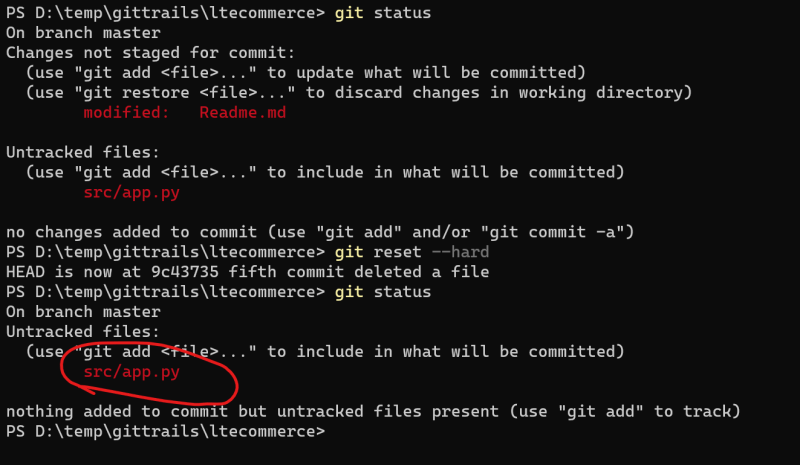
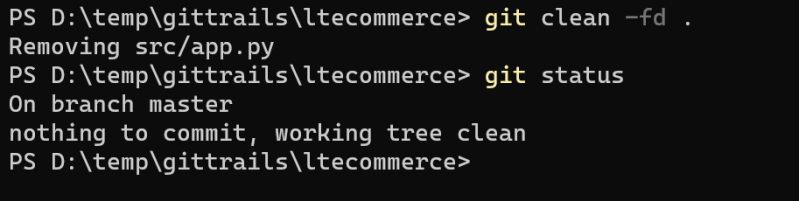
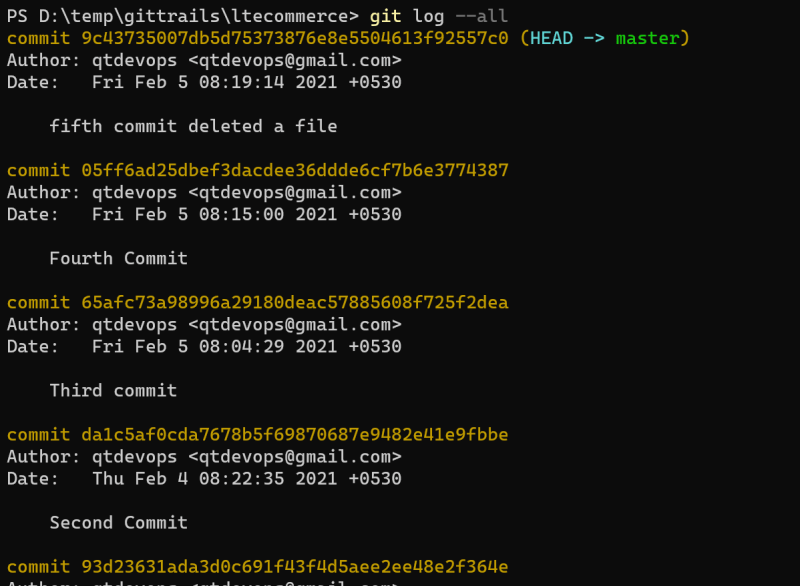
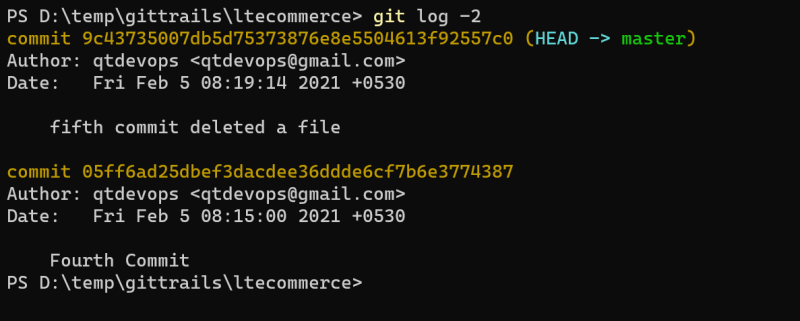
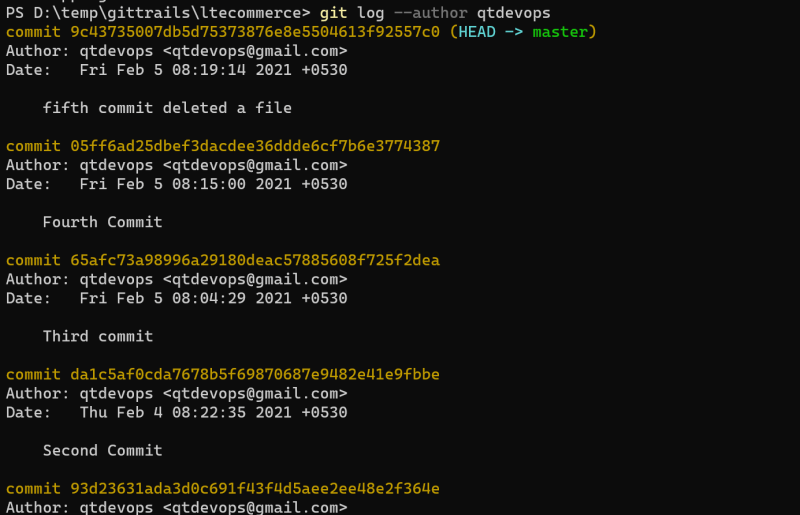
* Create a local repository
  + Create any empty folder 
  + Open this folder in any terminal (Powershell, Git Bash)  
  + Now initialize the repository by executing git init 
  + When we initialize the repository a .git folder is generated 
* Three areas in git on your local system 
* Now lets make changes in working tree
  + Create a file in working tree 
  + Now lets ask git what is the status by executing git status 
  + Understanding add and commit 
  + Now lets add the change of working tree to index area 
  + Now lets commit the changes to local repo. But before we commit the changes, commit needs
    - email id
    - username
    - message
  + Email id and username can be configure once per system, let do it Preview
  + Message is given for every commit git commit -m <message of commit>
  + Now lets commit our change from index area to local repository 
  + We had commited the changes from working tree to local repo via index area (staging area). Now lets see what the git status is 
  + Now lets see the history of the repository by executing git log 
* Now lets make one more change by adding some text to Readme.md

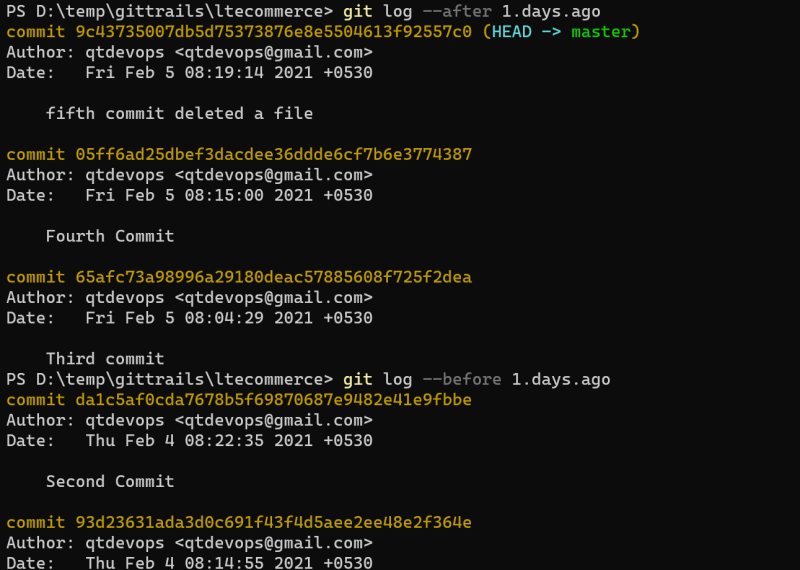
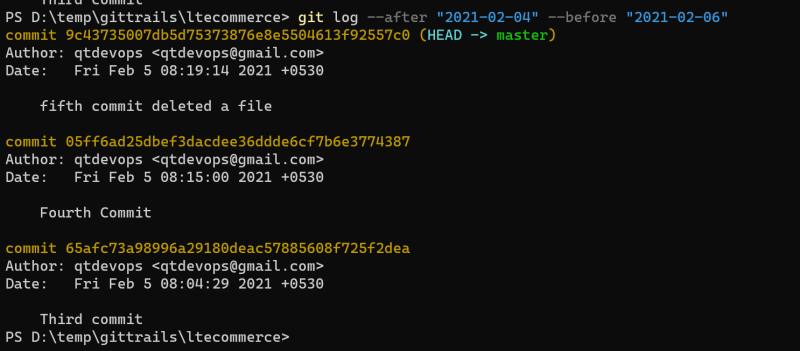


* Lets see what git status tells 
* Now lets add this change to indexing/staging area 
* Now lets commit the change from staging area to local repo 
* In GIT, Head is pointer to commit id. By default HEAD points to the latest commit. We can also travel back to older commits
* Note: Try using any cheatsheat of your choice to get used to commands [Refer Here](https://www.google.com/search?q=git+cheat+sheet&rlz=1C1GCEA_enIN920IN920&oq=git+cheat&aqs=chrome.0.0i433j69i57j0l6.2420j0j7&sourceid=chrome&ie=UTF-8)
* In Git to make changes we move the changes from one phase to other, So we might see some of documents referring git as phased commit.
* So now lets make some more changes by create two empty folders in working tree 
* Now lets execute git status 
* For git, changes are considered at file level, empty folders are not considered as changes in git.

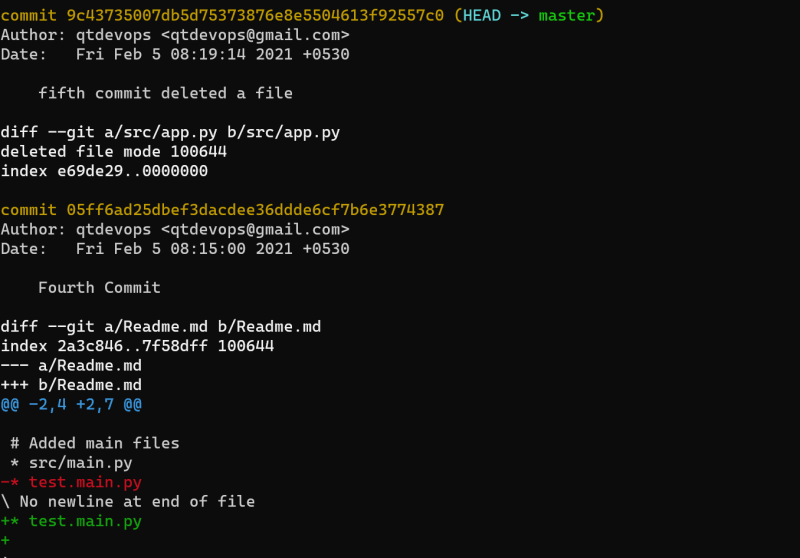
## Git contd.. 05/02/2021

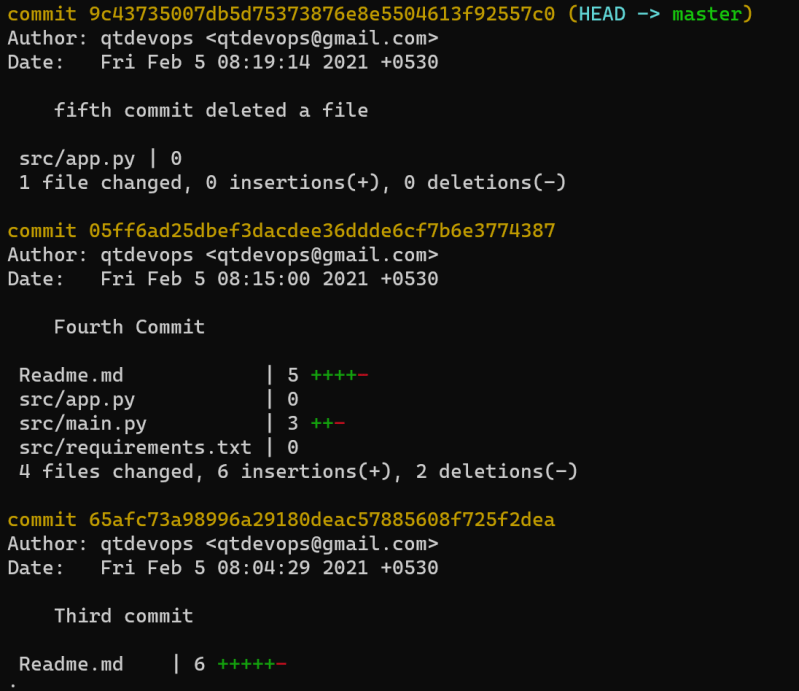
### Adding multiple changes to index area

* In Git we can deal with multiple changes at one time and add all of those changes to the staging/index area.
* Lets create two new files in folders and make changes to existing file and execute status command 
* For git
  + untracked file => This was not present in the local repo and you have created a new file
  + modified => Made changes to existing files
  + If the new files are part of some folder, then git will show folder in the status but actually will track files.
* Now lets add the changes 
* Now lets commit the changes to the local repository 
* Now lets create two more files in src app.py and requirements.txt 
* Now make changes in existing file (Readme.md) and execute git status 
* Now if we want to add only modified files not untracked files to index/staging area 
* Now lets make some more changes in main.py of src 
* Now lets navigate to test folder
  + here git add . will not workout as changes are not in test folder, lets add all the files to the staging area using git add -A  
* Lets do a change by deleting a file src/app.py 
* Now lets add these changes to repo via staging area 
* Now execute git log 
* Now lets make changes in 3 files and add that to staging area  
* Now lets try to reset the change in test/main.py and src/main.py 
* Now if we want to remove the changes in test/main.py and src/main.py in the working tree as well 
* If you want to remove all the changes in index area and working tree with one command 
* Now lets create a new set of changes with untracked files, modified files 
* Now to remove the untracked files 
* Exercise: Findout the difference b/w git reset –hard and git reset –soft
* Git Log Variants
  + Display All commits 
  + View n most recent commits git log -n 
  + Filter commits by author 
  + Filter commits by date
* git log --before <date>
* git log --after <date>

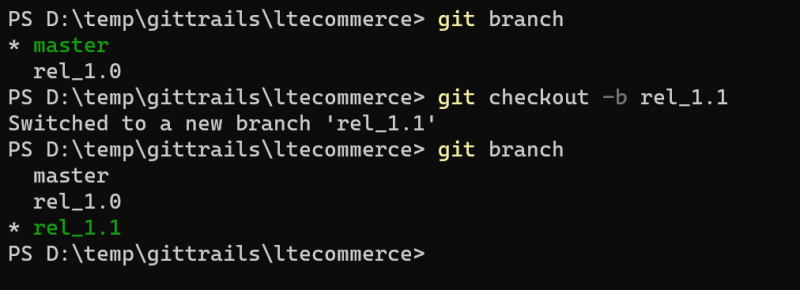
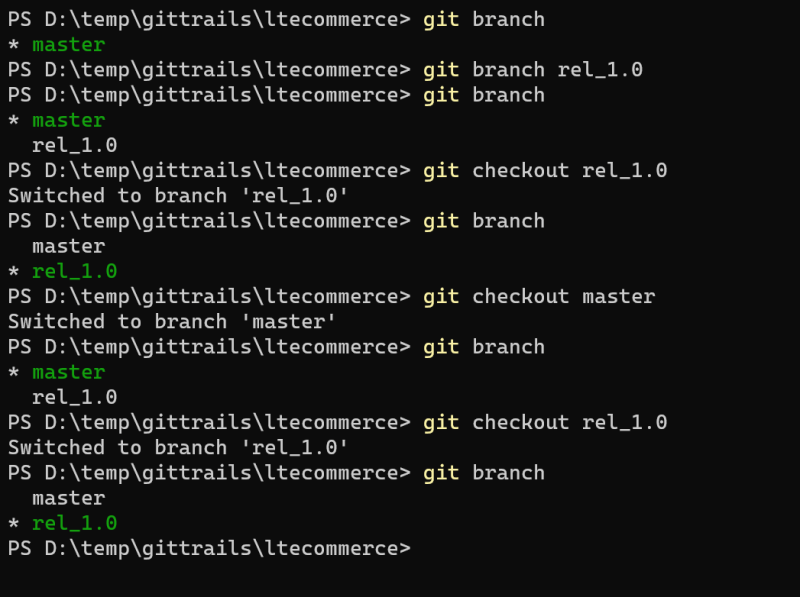
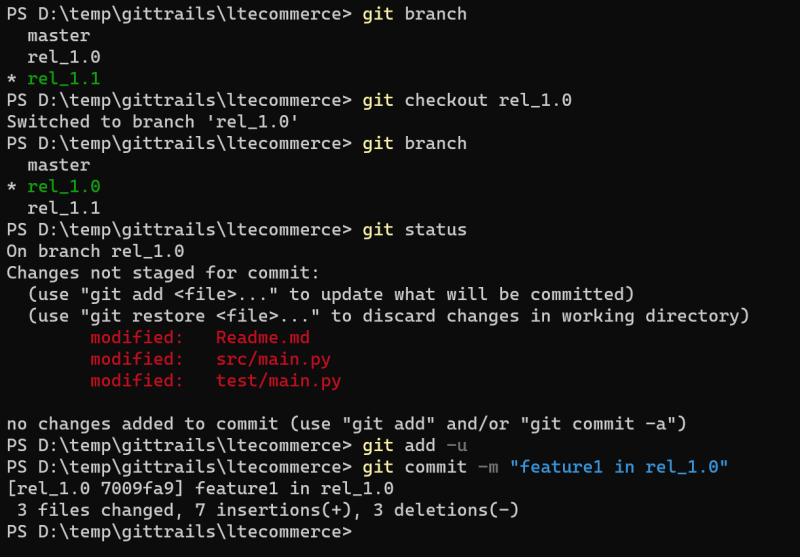
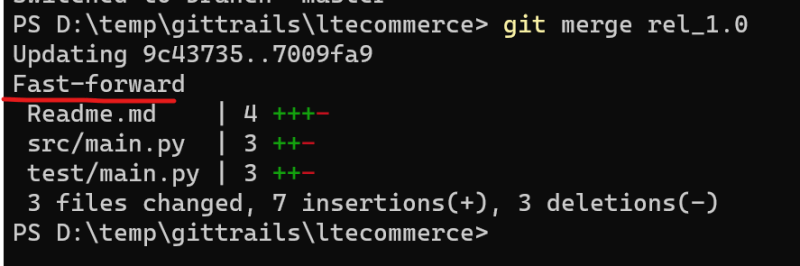
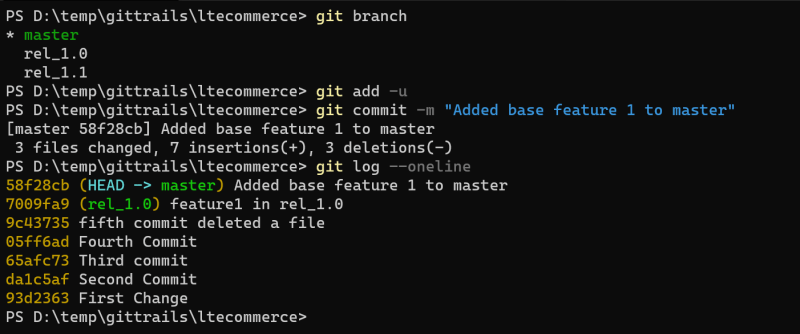
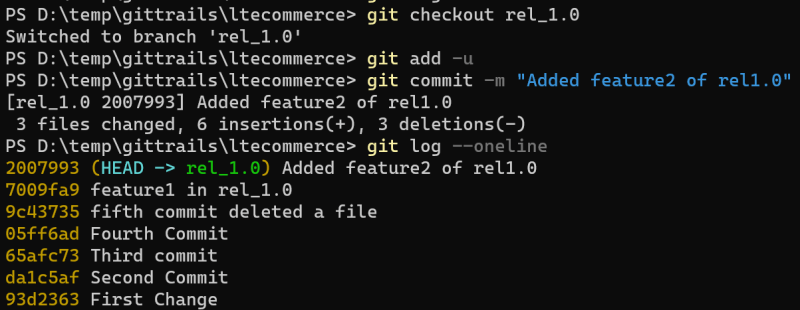
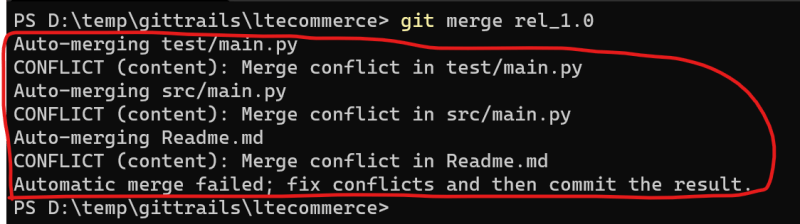
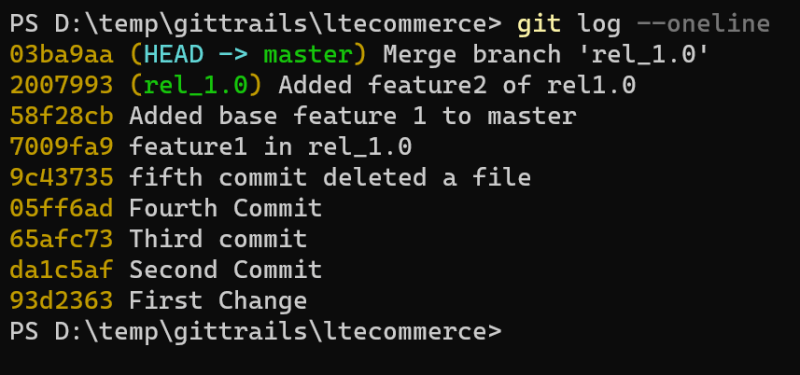
* + Viewing all the changes done in each commit git log -p

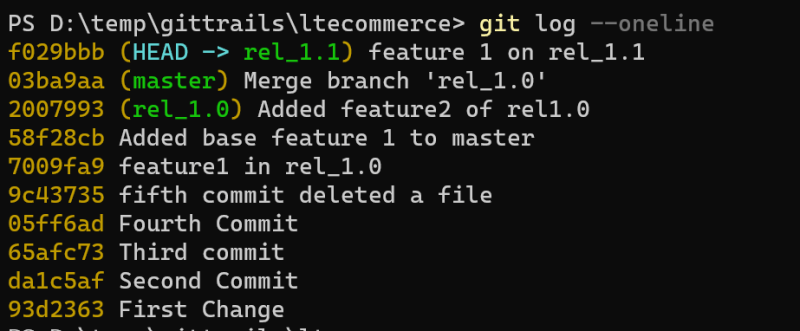


* + To view summary of each commit git log --stat 

## Git Contd.. – 07/02/2021

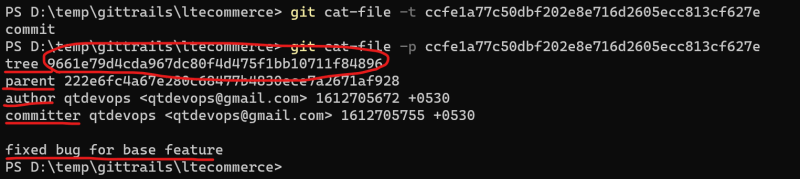
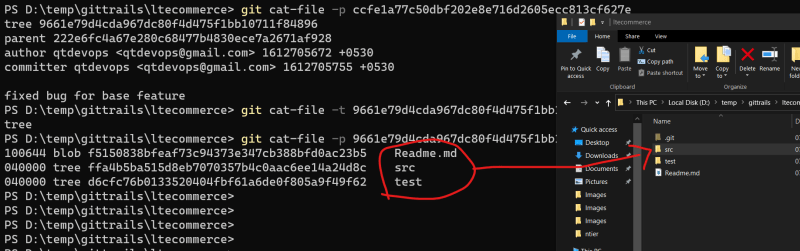
### Scenario: Understanding Branches

* There will be many cases where parallel tracks of development will happpen on your projects
* Git branches enable you to create parallel tracks of development. In Git there is already a default branch which is referred as master
* Now lets create two more branches
  + rel\_1.0 
  + rel\_1.1 
* Now lets make a change in rel\_1.0 and commit the changes to local repo 
* Now lets make a change in rel\_1.1 and commit the changes to local repo
* Branch will point to the latest commit on that branch and head will be pointing towards branch which points to latest commit
* Now we need changes of rel\_1.0 branch to be in master.
  + Fast forward merge 
* Lets make changes in master branch and commit those changes and also do some changes on rel\_1.0 branch  
* Now we need to merge the changes from rel\_1.0 to master
  + Merges might lead to conflicts and that needs to be resolved  
* Git Rebase
  + [Refer Here](https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase#:~:text=From%20a%20content%20perspective%2C%20rebasing,them%20to%20the%20specified%20base.) for documenation
  + Lets rebase master onto rel\_1.1
* git checkout rel\_1.1
* git rebase master
* # fix merge conflicts
* # add the changes usign git add
* # then continue the rebase git rebase --continue

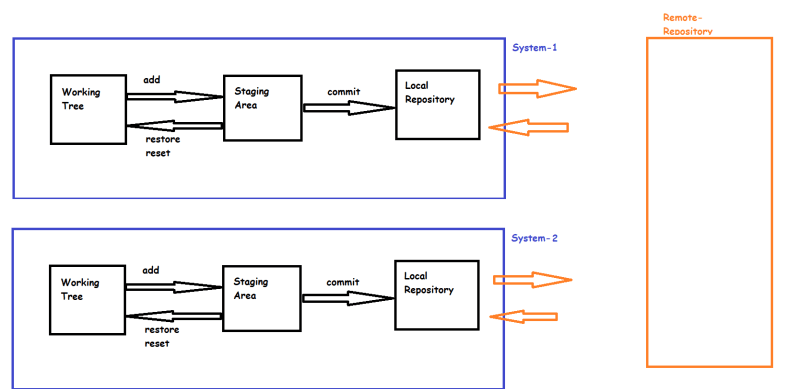
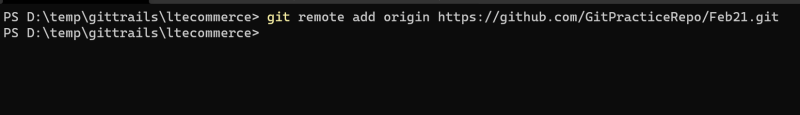
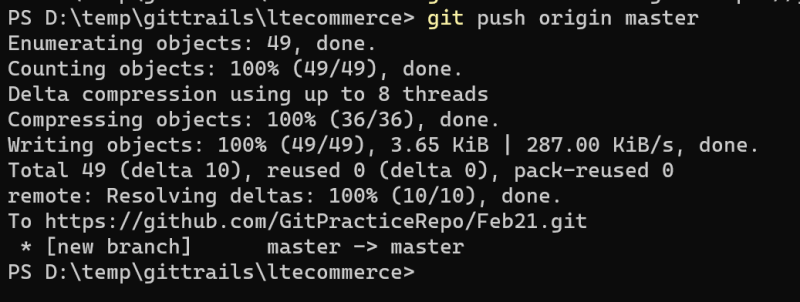
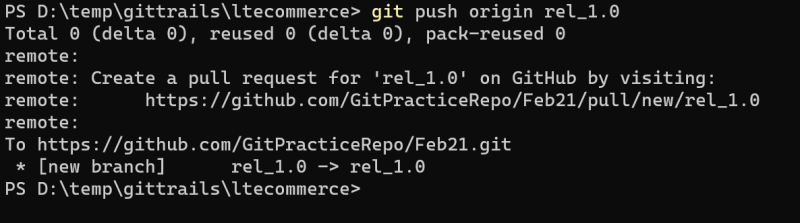
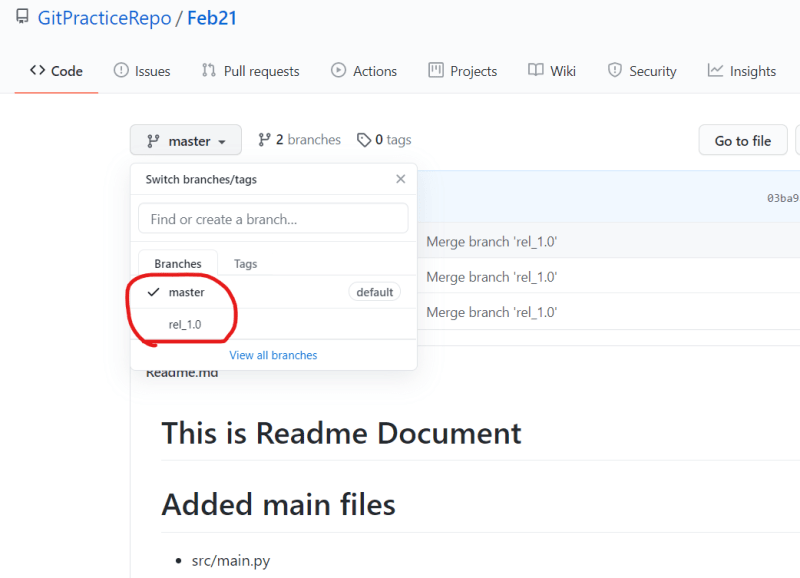
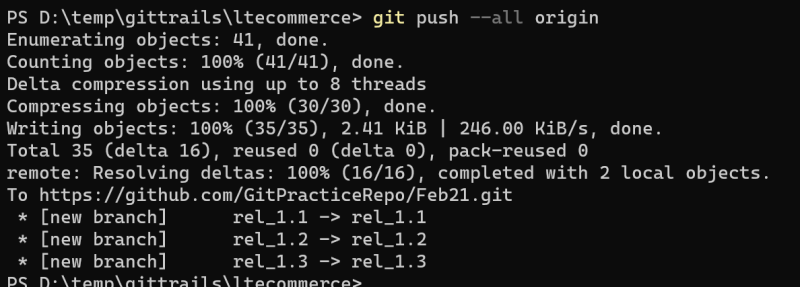
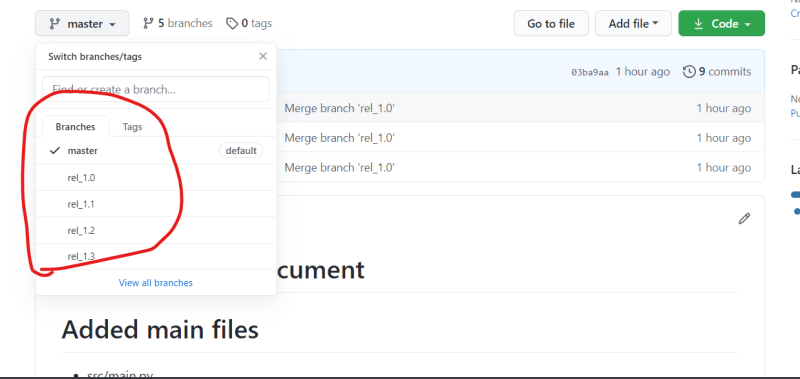
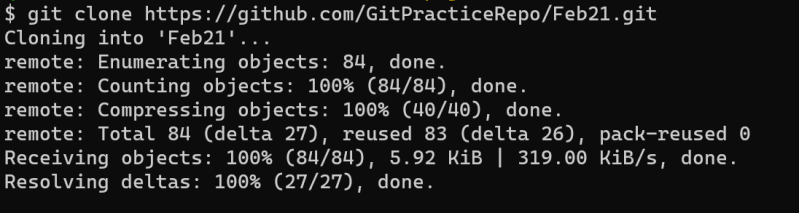
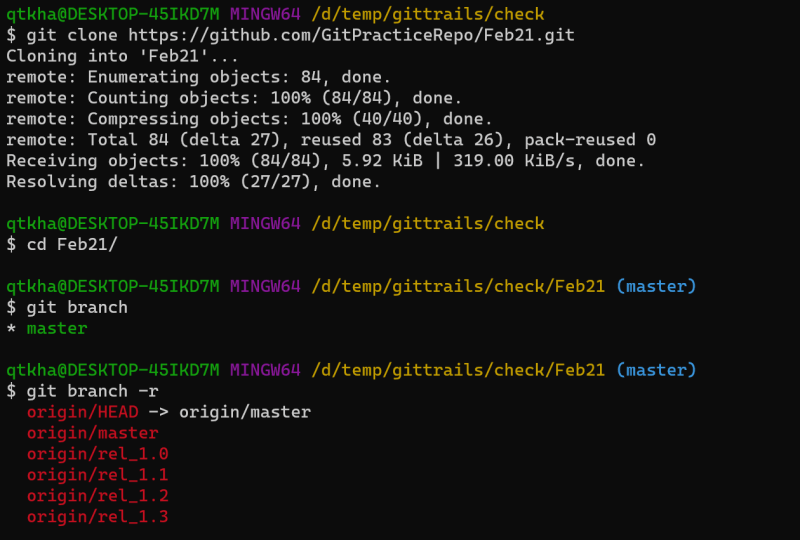
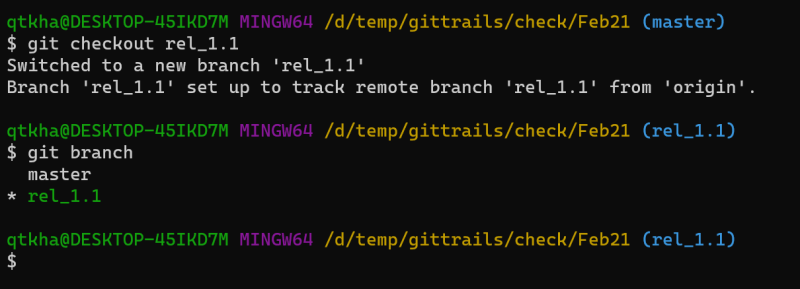
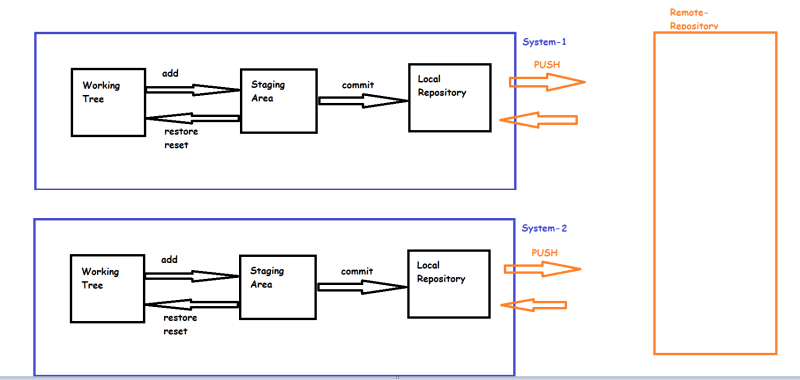


* In git if you want to pick certain commits and add it to the branch then that is referred as cherry-picking
* Now lets create two branches from master
  + rel\_1.2
  + rel\_1.3
* [Refer Here](https://www.atlassian.com/git/tutorials/cherry-pick#:~:text=git%20cherry%2Dpick%20is%20a,to%20the%20current%20working%20HEAD.&text=For%20example%2C%20say%20a%20commit,to%20where%20it%20should%20belong.) for cherry-pick documentation
* Now we have understood three ways bringing changes from one branch to other
  + merge
  + rebase
  + cherry-pick

How Git Works

* To understand how git works, we need to know about
  + Hashing (SHA1):
    - This is transformation of string of characters into a shorted fixed length value
    - Two text with same value will have same hash
* Git can be reffered as a stupid content tracker.
* In Git every commit id is generated by calculating hash of
  + Changes
  + Author
  + Date time
  + Message
* Every commit will have a parent commit 
* In Git Commit tree stands for folder and binary large object (blob) stands for file  

Fourth Area of git

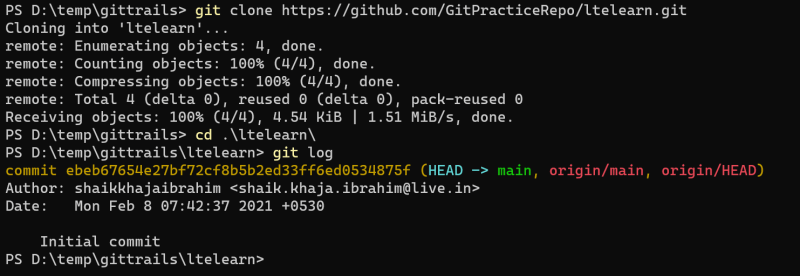
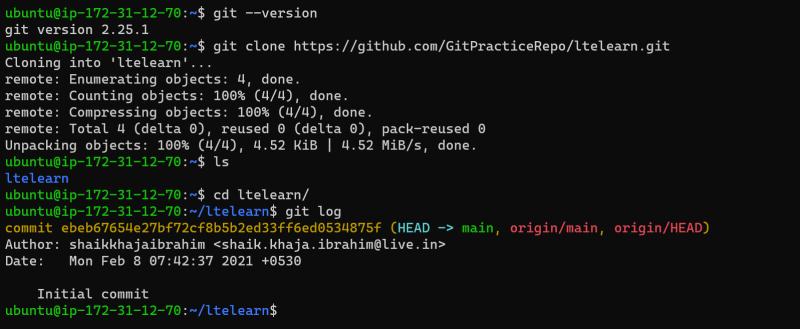
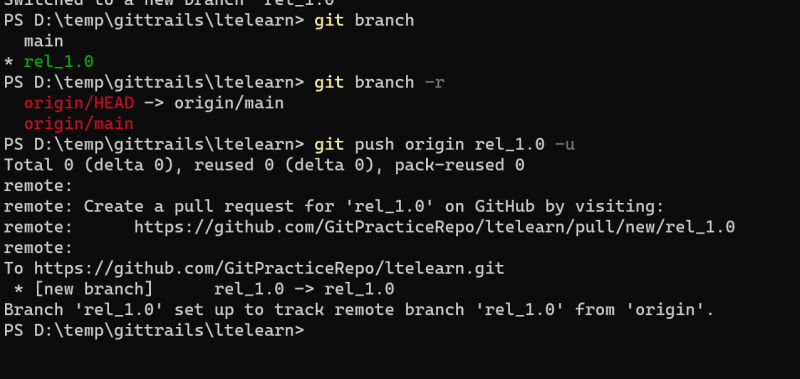
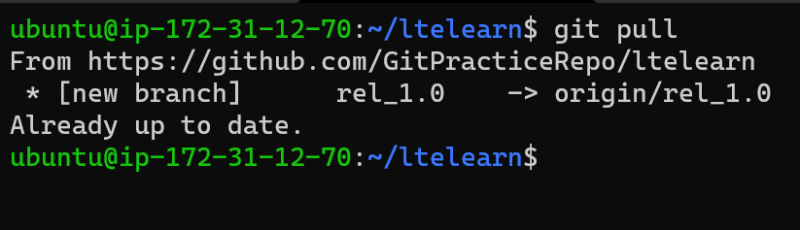
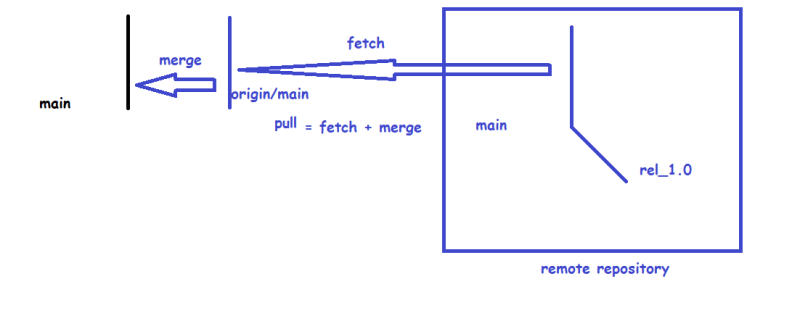
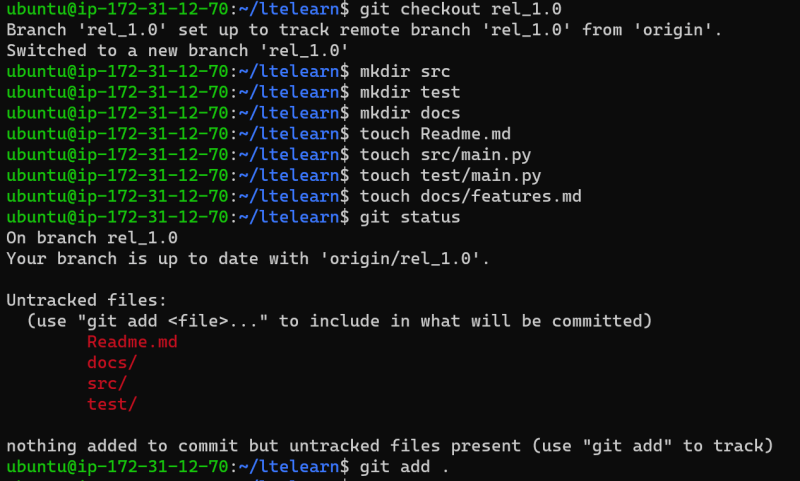
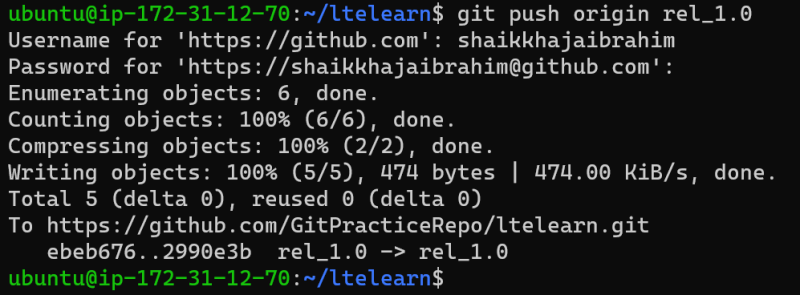
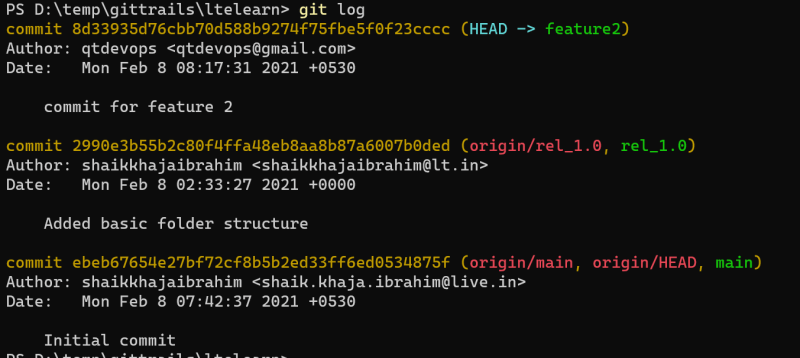
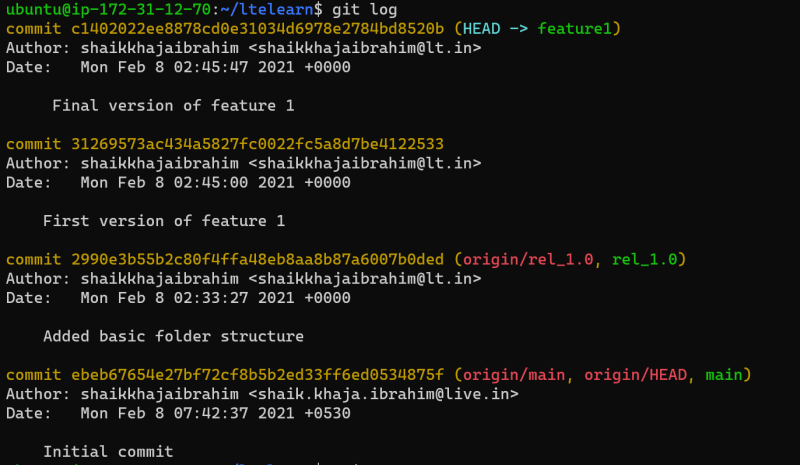
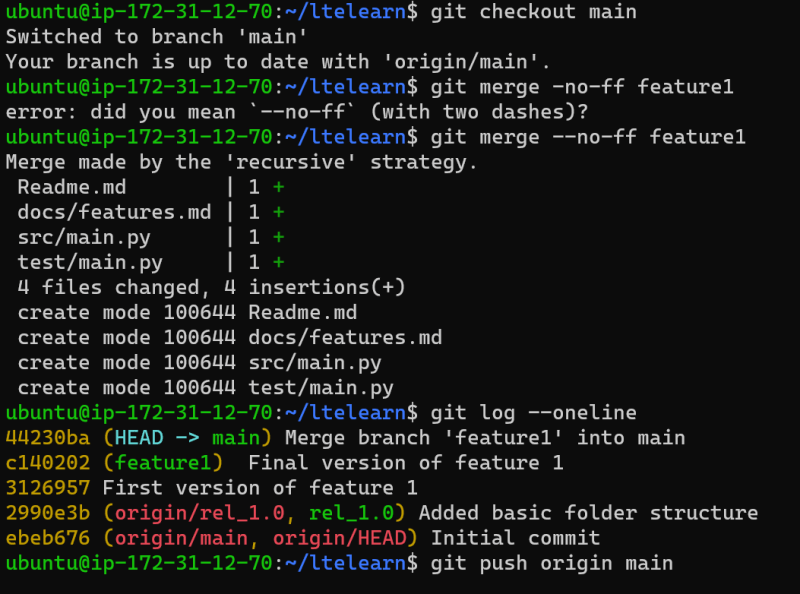
* Fourth area of git is referred as remote repository, which is used to collobarate work done by multiple developers in an orginization 
* Git remote repository contains the same .git folder and in addition to that it will have
  + connectivity options (for developers to connect)
* To have this on git remote repository some daemon will be running.
* There are many ways to configure git remote repository
  + We can host git remote repository on some shared folder in your organization
  + We can use some git remote repository softwares
    - Gitolite
    - GitLab
  + We can use some hosted git remote repository options
    - GitHub
    - BitBucket Cloud
    - GitLab
    - Azure Source repos
    - AWS Code Commit
* We can have a local repository which is connected to multiple remote repositories
* Just like master was a default branch, the default remote repository also will have a name *origin*
* So now lets create a new repository in GitHub and send all the changes which we have done in the class to that repository
  + Add remote repository to local repo 
  + Sending the changes from local repo to remote repo is referred as push and the command for pushing is git push <name-of-remote> <name-of-branch>   
  + Lets try to push all the branches to remote repo  
* How to get the copy of the code from existing remote repository
  + This operation is referred as clone.  
  + Now lets checkout to rel\_1.1 
* To push the changes to remote repository, we need permissions 

Next Steps

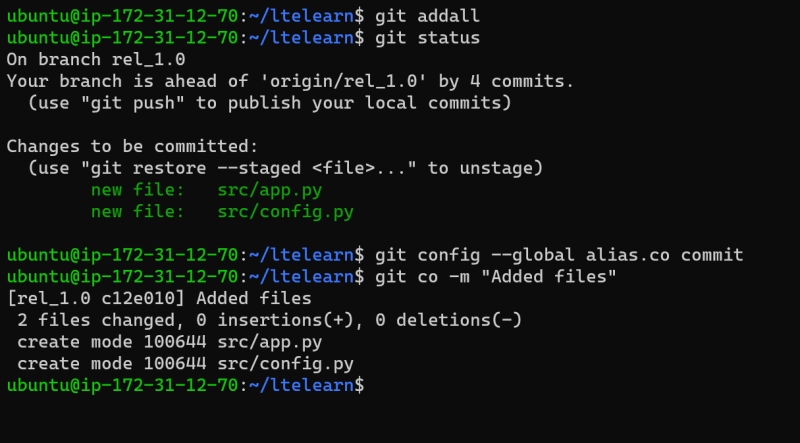
* Understanding multiple users workflow
* Understanding recieving latest changes from remote repository
* Understanding git tags
* How to make changes to the existing commits

### Multiple Users Workflow – 08/02/2021

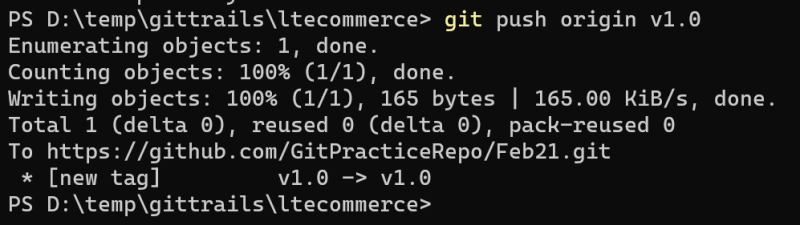
#### Scenario:

* Your Organization has a remote repository configured on GitHub
* Two developers are working on features and they push the changes to git Hub  
* Now developers are working for release 1.0
  + Lets create a branch called as rel\_1.0 and push the branch to github   
* Now developer 1 is working on linux machine machine
  + He is working on to create basic folder structure and he commits the changes 
  + Now configure username and email and commit the changes to local repo
  + Now lets push the changes to remote repository 
* Developer 2 who is working on windows machine needs to get the changes submitted to start working
* Now Developer 1 will be working on feature 1 and Developer 2 will be working on feature 2
  + To do this, lets assume organization is following feature branch approach
  + Feature branch approach is where developers create a branch locally for every feature and once the finish developing the changes, they will merge the changes to the rel\_1.0 branch and push the changes  
* Now developer1 has merged the changes of feature1 to rel\_1.0 branch and pushed the changes to the repository 
* Now developer 2 wants to merge the change of feature 2 to rel\_1.0 branch.
* To ignore some files or folders we create a file called .gitignore in the root folder and specify which folders or files need to be excluded from git
* [Refer Here](https://github.com/GitPracticeRepo/ltelearn/blob/feature1/.gitignore)
* To generate git ignore file for the tools which you use [Refer Here](https://www.toptal.com/developers/gitignore)

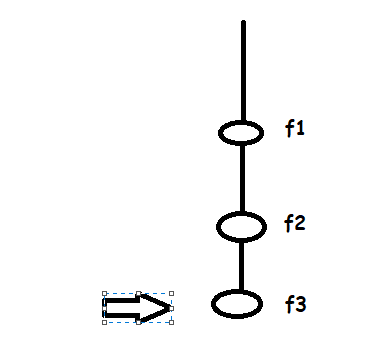
Git configuration

* This is config command to set some configuration values
* In Git we have different config levels and files
  + –local: This is present in the .git/config
  + –global: This applies to a particular os user ~/.gitconfig
  + –system: This a system level configuration applied to all the users on the machine $(installpath)/etc/gitconfig
* [Refer Here](https://git-scm.com/docs/git-config) for configuration values from official docs and [Refer Here](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-config) for attlasian documentation
* Git Alias 

Git Tags

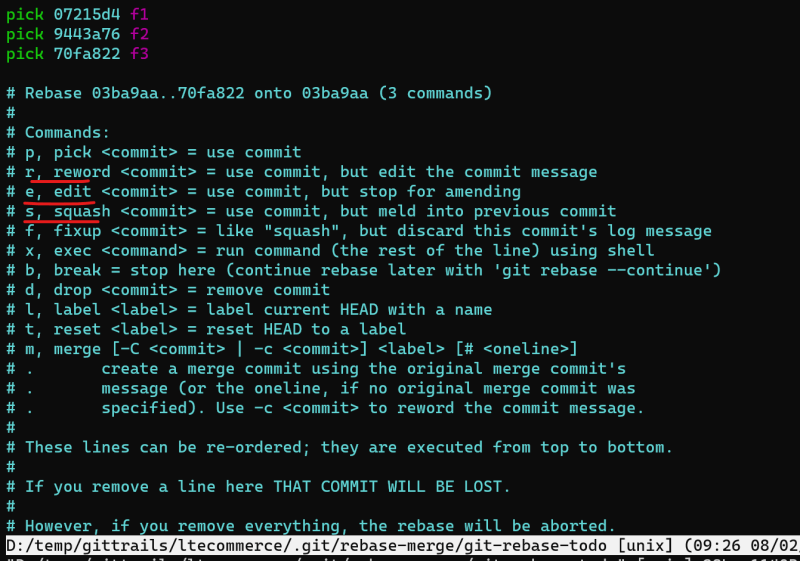
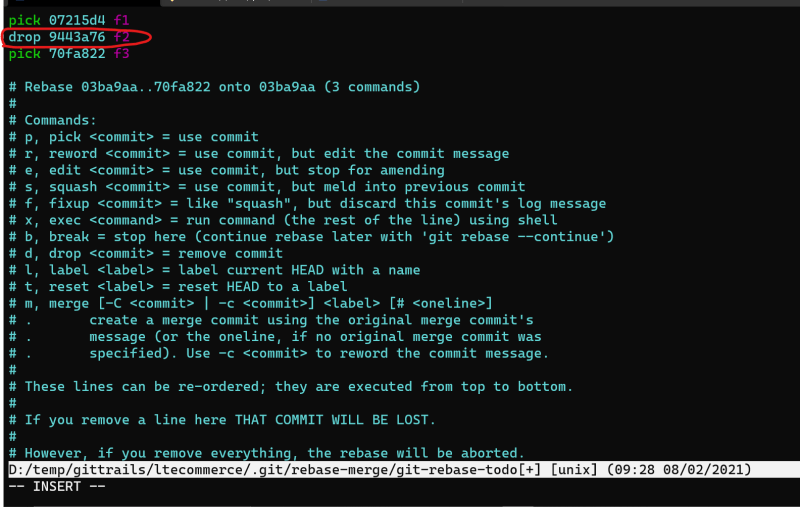
* Tag refers to the specific commit in git history
* Git supports two kinds of tags
  + Lightweight tags
  + Annotated Tags
* Annotated tags differ in amount of metadata they store  
* [Refer Here](https://git-scm.com/book/en/v2/Git-Basics-Tagging) for the official docs and [Refer Here](https://www.atlassian.com/git/tutorials/inspecting-a-repository/git-tag) for attlasian tutorial

Changing the previous commits

* Lets assume you have 3 commits f1,f2,f3 
* Now if you want to delete the f2 commit

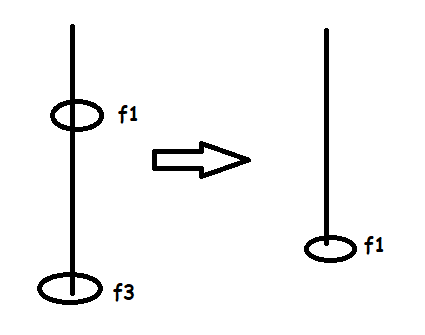
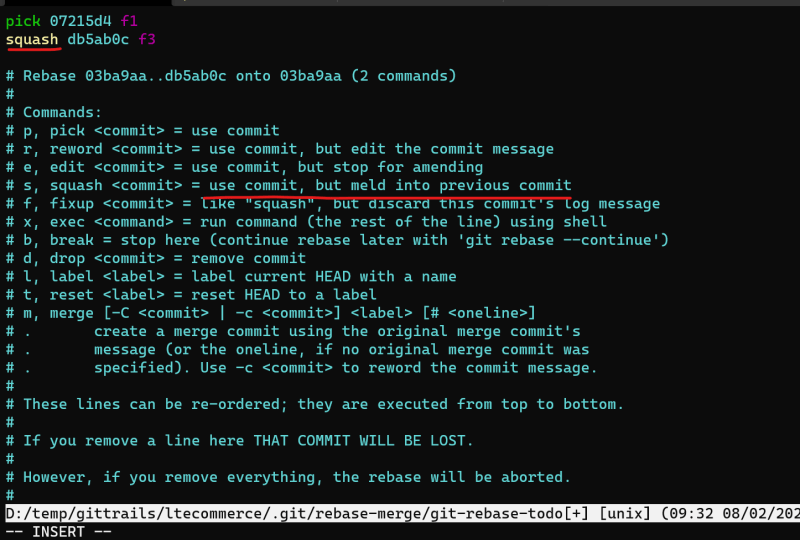
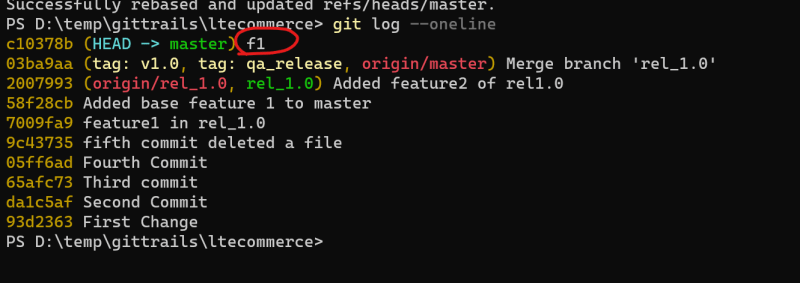
git rebase -i HEAD~3

# fix merge conflicts if any and continue rebase

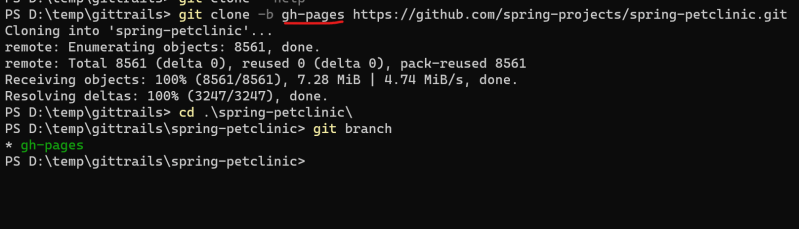
 

* Now lets try to combine f1 and f3 as one commit

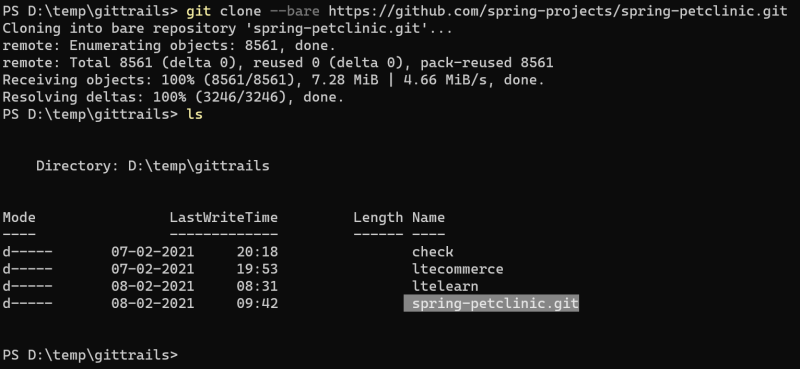
git rebase -i HEAD~2

Cloning a Particular Branch on Git repo

* Clone only gh-pages branch from spring petclinic [Refer Here](https://github.com/spring-projects/spring-petclinic.git) 

Git bare repositories

* Generally when we clone repositories we get the working tree and .git folder.
* But on git remote repositories (i.e. git on servers) we might not need working tree as no one will be working there, there we can clone bare repositories
* In bare repositories you get only .git folder 

Next Steps

* We will learn the git communication protocols
* GitHub/BitBucket/Git Pull request
* Git branching strategy
* We will learn how to build the code and start using jenkins and then visit the leftover git topics