Complete GIT Crash Course

Learn Git from Scratch

Trainer – Haradhan Pal

Agenda

- Introduction to Git
- Features, Advantages and Disadvantages of Git
- Version Control System and it’s Benefit
- Types of Version Control Systems
- Install Git in your System
- Components of Git
- Basic Git commands
- Git and GitHub Synchronization
- Create new Branch
- Push, Pull and Merge Code
- Revert any changes (Commit) with Commit Id

Git Introduction

- Git is a free, open-source, modern and widely used distributed version control system (DVCS) in the world. DVCSs allow full access to every file, branch, and iteration of a project, and allows every user access to a full and self-contained history of all changes. Unlike once popular centralized version control systems, DVCSs like Git don't need a constant connection to a central repository. Developers can work anywhere and collaborate asynchronously from any time zone. It was created by Linus Torvalds in 2005. This tool is a version control system that was initially developed to work with several developers on the Linux kernel. This basically means that Git is a content tracker. So, Git can be used to store content — and it is mostly used to store code because of the other features it provides. It is developed to manage projects with high speed and efficiency. The version control system allows us to monitor and work together with our team members at the same workspace. Real life projects generally have multiple developers working in parallel. So, they need a version control system like Git to make sure that there are no code conflicts between them. Also, the requirements in such projects change often. So, a version control system allows developers to revert and go back to an older version of their code.
- Without version control, team members are subject to redundant tasks, slower timelines, and multiple copies of a single project. To eliminate unnecessary work, Git and other VCSs give each contributor a unified and consistent view of a project, surfacing work that's already in progress. Seeing a transparent history of changes, who made them, and how they contribute to the development of a project helps team members stay aligned while working independently.
- According to the latest Stack Overflow developer survey, more than 70 percent of developers use Git, making it the most-used VCS in the world. Git is commonly used for both open source and commercial software development, with significant benefits for individuals, teams and businesses.
- Git lets developers see the entire timeline of their changes, decisions, and progression of any project in one place. From the moment they access the history of a project, the developer has all the context they need to understand it and start contributing.
- Businesses using Git can break down communication barriers between teams and keep them focused on doing their best work. Plus, Git makes it possible to align experts across a business to collaborate on major projects.

Features of Git

- Open Source: Git is an open-source and widely used free tool and no licensee cost
- Scalable: Git is scalable, which means when the number of users increases, the Git can easily handle such situations.
- Distributed: Git is distributed which means that instead of switching the project to another machine, user can create a "clone" of the entire repository. Also, instead of just having one central repository that user send changes to, every user has their own repository that contains the entire commit history of the project. User do not need to connect to the remote repository; the change is just stored on our local repository. If necessary, user can push these changes to a remote repository.
- Security: Git is very secure. It uses the SHA1 (Secure Hash Function) to name and identify objects within its repository. Files and commits are checked and retrieved by its checksum at the time of checkout. It stores its history in such a way that the ID of commits depends upon the complete development history leading up to that commit. Once it is published, one cannot make changes to its old version.
- Speed: Git is very fast, so it can complete all the tasks in few seconds. Most of the git operations are done on the local repository, so it provides a huge speed. Also, a centralized version control system continually communicates with a server somewhere. Fetching version history from a locally stored repository is much faster than fetching it from the remote server. The core part of Git is written in C, which ignores runtime overheads associated with other high-level languages.
- Supports non-linear development: Git supports seamless branching and merging, which helps in visualizing and navigating a non-linear development. A branch in Git represents a single commit. User can construct the full branch structure with the help of its parental commit.
- Data Assurance: The Git data model ensures the cryptographic integrity of every unit of user project. It provides a unique commit ID to every commit through a SHA algorithm. We can retrieve and update the commit-by-commit ID. Most of the centralized version control systems do not provide such integrity by default.
- Staging Area: The Staging area is also a unique functionality of Git. It can be considered as a preview of our next commit, moreover, an intermediate area where commits can be formatted and reviewed before completion. When user make a commit, Git takes changes that are in the staging area and make them as a new commit. User are allowed to add and remove changes from the staging area. The staging area can be considered as a place where Git stores the changes.
- Maintain the clean history: Git facilitates with Git Rebase; It is one of the most helpful features of Git. It fetches the latest commits from the master branch and puts user code on top of that. Thus, it maintains a clean history of the project.
- Branching and Merging: Branching and merging are the great features of Git, which makes it different from the other SCM tools. Git allows the creation of multiple branches without affecting each other. We can perform tasks like creation, deletion, and merging on branches, and these tasks take a few seconds only.

Advantages of Git

- Performance: Git performs very strongly and reliably when compared to other version control systems. New code changes can be easily committed, version branches can be effortlessly compared and merged, and code can also be optimized to perform better.
- Offline Working: One of the most important benefits of Git is that it supports offline working. If we are facing internet connectivity issues, it will not affect our work. In Git, user can do almost everything locally. Comparatively, other CVS like SVN is limited and prefer the connection with the central repository.
- Quality open-source project: Git is a widely supported open-source project with over ten years of operational history. People maintaining the project are very well matured and possess a long-term vision to meet the long-term needs of users by releasing staged upgrades at regular intervals of time to improve functionality as well as usability. Quality of open-source software made available on Git is heavily scrutinized a countless number of times and businesses today depend heavily on Git code quality.
- Undo Mistakes: One additional benefit of Git is we can Undo mistakes. Sometimes the undo can be a savior option for us. Git provides the undo option for almost everything.
- Track the Changes: Git facilitates with some exciting features such as Diff, Log, and Status, which allows us to track changes so we can check the status, compare our files or branches.
- Wide acceptance: Git offers the type of performance, functionality, security, and flexibility that most developers and teams need to develop their projects. When compared to other VCS Git is the most widely accepted system owing to its universally accepted usability and performance standards.
- Community: Git is very popular, widely used, and accepted as a standard version control system by the vast majority within the developer's community. It's much easier to leverage 3rd-party libraries and encourage other developers to fork user open-source code using Git. The sheer number of Git users make it easy to resolve issues and seek outside help using online forums.

Disadvantages of Git

- GIT requires technical excellence, and it is slower on windows. They have tedious command lines to input and doesn’t track renames.
- It lacks window support and doesn’t track empty folders.
- There is no built-in access control, and it fails with the presence of files having binary data. It starts processing every work slowly. Any file which doesn’t support textual data is not compatible with this technology.
- It has poor GUI and usability and take a lot of resources which slows down the performance.
- GIT needs multiple branches to support parallel developments used by the developers.

Version Control System

- Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems (VCS) are software tools that help software teams manage changes to source code over time. As development environments have accelerated, version control systems help software teams work faster and smarter. They are especially useful for DevOps teams since they help them to reduce development time and increase successful deployments.
- Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.
- Software developers working in teams are continually writing new source code and changing existing source code. The code for a project, app or software component is typically organized in a folder structure or "file tree". One developer on the team may be working on a new feature while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree.
- Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting. Changes made in one part of the software can be incompatible with those made by another developer working at the same time. This problem should be discovered and solved in an orderly manner without blocking the work of the rest of the team. Further, in all software development, any change can introduce new bugs on its own and new software can't be trusted until it's tested. So, testing and development proceed together until a new version is ready.
- Good version control software supports a developer's preferred workflow without imposing one way of working. Ideally it also works on any platform, rather than dictate what operating system or tool chain developers must use. Great version control systems facilitate a smooth and continuous flow of changes to the code rather than the frustrating and clumsy mechanism of file locking - giving the green light to one developer at the expense of blocking the progress of others.
- Software teams that do not use any form of version control often run into problems like not knowing which changes that have been made are available to users or the creation of incompatible changes between two unrelated pieces of work that must then be painstakingly untangled and reworked. If you're a developer who has never used version control you may have added versions to your files, perhaps with suffixes like "final" or "latest" and then had to later deal with a new final version. Perhaps you've commented out code blocks because you want to disable certain functionality without deleting the code, fearing that there may be a use for it later. Version control is a way out of these problems.
- Version control software is an essential part of the every-day of the modern software team's professional practices. Individual software developers who are accustomed to working with a capable version control system in their teams typically recognize the incredible value version control also gives them even on small solo projects. Once accustomed to the powerful benefits of version control systems, many developers wouldn't consider working without it even for non-software projects.

Benefits of Version Control System

- Enhances the project development speed by providing efficient collaboration.
- Leverages the productivity, expedite product delivery, and skills of the employees through better communication and assistance.
- Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change.
- All the previous versions and variants are neatly packed up inside the VCS. You can request any version at any time as per your requirement and you'll have a snapshot of the complete project right at hand.
- Employees or contributor of the project can contribute from anywhere irrespective of the different geographical locations through this VCS, for each different contributor of the project a different working copy is maintained and not merged to the main file unless the working copy is validated. A most popular example is Git, Helix core, Microsoft TFS.
- A distributed VCS like Git allows all the team members to have a complete history of the project so if there is a breakdown in the central server you can use any of your teammate's local Git repository.
- Informs us about Who, What, When Why changes have been made.

Types of Version Control Systems

- Local Version Control Systems: It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.
- Centralized Version Control Systems: Centralized version control systems contain just one repository, and each user gets their own working copy. You need to commit to reflecting your changes in the repository. It is possible for others to see your changes by updating. The benefit of CVCS (Centralized Version Control Systems) makes collaboration amongst developers along with providing an insight to a certain extent on what everyone else is doing on the project. It allows administrators to fine-grained control over who can do what. It has some downsides as well which led to the development of DVS. The most obvious is the single point of failure that the centralized repository represents if it goes down during that period collaboration and saving versioned changes is not possible. What if the hard disk of the central database becomes corrupted, and proper backups haven't been kept? You lose absolutely everything.
- Distributed Version Control Systems: Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository, and you need to push them in order to make them visible on the central repository. Similarly, When you update, you do not get other's changes unless you have first pulled those changes into your repository.

Examples of Version Control Systems

- There are several types of version control systems that teams use today. Some are centralized. Some are distributed. Here are a few of the most popular types of VCS:

  - Helix Core (Perforce)
  - Git
  - SVN
  - ClearCase
  - Mercurial
  - TFS

Difference between Git and SVN

Parameters of Comparison	**Git**	**SVN**
Branching	The branches in Git are easy to edit or delete. This process does not result in conflicts.	The branches in the directory are a pain to work on. This difficulty is because this results in three conflicts.
Storage	Large binary files are difficult to store in these systems.	Large binary files are easy to store, and they do not take as much place.
User Friendly	Git has a much more complicated interface and functionality.	SVN has a more straightforward interface that one can use comfortably in a short period.
Speed	Changes can take place at a faster rate because of the easy branching.	Editing a file on SVN can be challenging as it has a more rigid directory.
Saving changes	A distributed system will include a local repository in which new files containing changes can be present.	A centralized system includes a central server only, and hence changes are seen in the original file directly.

Difference between Git and GitHub

- Git is a version control system of distributed nature that is used to track changes in source code during software development. It aids in coordinating work among programmers, but it can be used to track changes in any set of files. The main objectives of Git are speed, data integrity, and support for distributed, non-linear workflows.
- GitHub is a Git repository hosting service, plus it adds many of its own features. GitHub provides a Web-based graphical interface. It also provides access control and several collaboration features, basic task management tools for every project.

Install Git in your System

- Navigate to https://git-scm.com/downloads to install Git in your local desktop or laptop.
- Download the appropriate Git file based on your Operating System.
- Open command prompt, enter "git" and press enter to check if it has been successfully installed in your system.
- Enter git --version to check the version of Git installed in your system
- Enter git --help to get all help option

Git Rrepository

- Repository in Git is a place where Git stores all the files. Git can store the files either on the local repository or on the remote repository.
- It is a virtual storage of your project where you keep all the resources/files of the project along with a special folder called.git.

Language used in Git

Git is a fast and reliable version control system, and the language that makes this possible is ‘C.’ Using C language reduces the overhead of run times, which are common in high-level languages.

Most popular Git hosting repositories

- GitHub
- GitLab
- BitBucket
- Beanstalk
- FogBugz
- Surround SCM
- Buddy

Components of Git

- Git comes with built-in GUI tools like git bash, git-gui, and gitk for committing and browsing. It also supports several third-party tools for users looking for platform-specific experience.

- **GitBash**: Git Bash is an application for the Windows environment. It is used as Git command line for windows. Git Bash provides an emulation layer for a Git command-line experience. Git package installer contains Bash, bash utilities, and Git on a Windows operating system. Bash is a standard default shell on Linux and mac OS. A shell is a terminal application which is used to create an interface with an operating system through commands. By default, Git Windows package contains the Git Bash tool.
- **Git GUI:** It is a powerful alternative to Git BASH. It offers a graphical version of the Git command line function, as well as comprehensive visual diff tools. User can access it by simply right click on a folder or location in windows explorer. Also, user can access it through the command line by typing command: git gui
- **Gitk:** It is a graphical history viewer tool. It's a robust GUI shell over git log and git grep. This tool is used to find something that happened in the past or visualize user project's history. Gitk can invoke from the command-line. Just change directory into a Git repository, and type: gitk

Basic Git commands

- To use Git, developers use specific commands to copy, create, change, and combine code. These commands can be executed directly from the command line or by using an application like GitHub Desktop or Git Kraken. Here are some common commands for using Git:
  - git init initializes a brand-new Git repository and begins tracking an existing directory. It adds a hidden subfolder within the existing directory that houses the internal data structure required for version control.
  - git clone creates a local copy of a project that already exists remotely. The clone includes all the project's files, history, and branches.
  - git add stages a change. Git tracks changes to a developer's codebase, but it's necessary to stage and take a snapshot of the changes to include them in the project's history. This command performs staging, the first part of that two-step process. Any changes that are staged will become a part of the next snapshot and a part of the project's history. Staging and committing separately gives developers complete control over the history of their project without changing how they code and work.

Basic Git commands - Contd

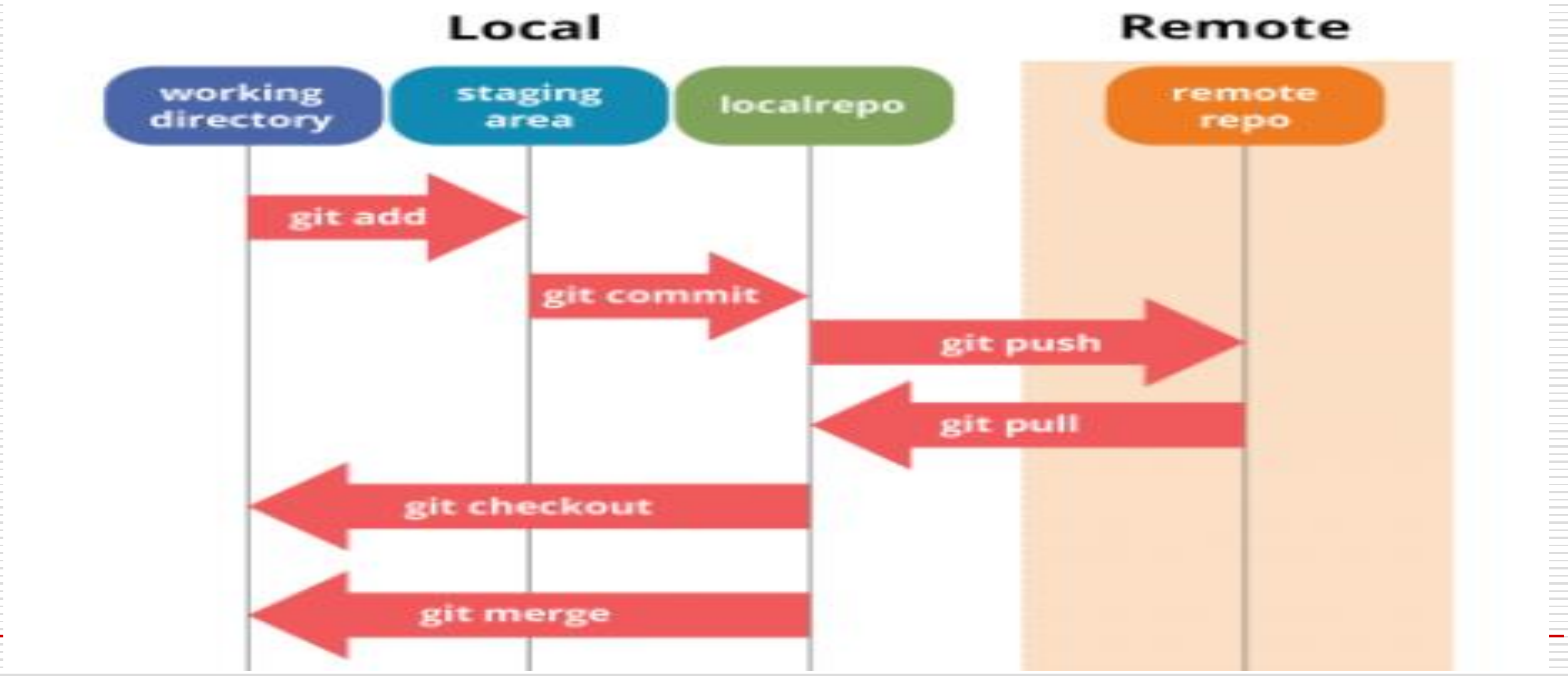
- git commit saves the snapshot to the project history and completes the change-tracking process. In short, a commit functions like taking a photo. Anything that’s been staged with git add will become a part of the snapshot with git commit.
- git status shows the status of changes as untracked, modified, or staged.
- git branch shows the branches being worked on locally.
- git merge merges lines of development together. This command is typically used to combine changes made on two distinct branches. For example, a developer would merge when they want to combine changes from a feature branch into the main branch for deployment.
- git pull updates the local line of development with updates from its remote counterpart. Developers use this command if a teammate has made commits to a branch on a remote, and they would like to reflect those changes in their local environment.
- git push updates the remote repository with any commits made locally to a branch.

Basic Git commands - Contd

- git checkout -b <branch_name> command create a Git branch and checkout in directly
- git branch -d<branch name> can be used to delete the specified branch
- git rebase apply a sequence of commits from distinct branches into a final commit
- git diff track the changes that have not been staged
- git log display the most recent commits and the status of the head
- git checkout switch between branches in a repository.
- git revert undo the changes for a particular commit
- git rm remove the files from the working tree and from the index

GIT Operations Architecture

- Git Bash and Git Shell are two different command line programs which allow user to interact with the underlying Git program. Bash is a Linux-based command line while Shell is a native Windows command line.
- Some of the basic operations in Git are:
- Initialize
- Add
- Commit
- Pull
- Push
- Refer the below architecture of Git about how these operations work with the Git repositories:



Create a GitHub Repository without README File

- Git isn't the same thing as GitHub. Git is a version-control system (i.e., a piece of software) that helps you keep track of your computer programs and files and the changes that are made to them over time. It also allows user to collaborate with their peers on a program, code, or file. GitHub and similar services (including GitLab and BitBucket) are websites that host a Git server program to hold user code.

- Steps to Create a Repository from GitHub:
- Navigate to the link: https://github.com/. Fill the sign-up form and click on “Sign up for Github”.
- Click on “Start a project” button.
- Enter any repository name and click on “Create Repository”. User can also give a description to their repository (optional). By default, a GitHub repository is public which means that anyone can view the contents of this repository whereas in a private repository, user can choose who can view the content. Also, private repository is a paid version. User can initialize the repository with a README file. This file contains the description of the file and once user check this box, this will be the first file inside user repository.
- So now user repository has been successfully created!

Access Repository from Git and Commit Changes

1. Open Git Bash terminal and enter the following command to create a folder named GitTraining

```
mkdir GitTraining
```

2. Change user terminal to the GitTraining directory with the below command:

```
cd GitTraining
```

3. Create a file named README.md and writes #GitGitHubTraining in the file with the below command:

```
echo "#GitGitHubTraining" >> README.md
```

4. Verify whether the file was created successfully and the content inside the file with the below command:

```
cat README.md
```

5. Initiate Git with the following command:

```
git init
```

6. Add the file in the directory:

```
git add README.md
```

Push the Changes to GitHub

7. Commit the changes. Commit can be thought of as a milestone. Every time user accomplish some work, they can write a Git commit to store that version of their file, so user can go back later and see what it looked like at that point in time. Whenever they make a change to their file, user create a new version of that file, different from the previous one.

```
git commit -m "first commit"
```

8. Go the main branch

```
git branch -M main
```

9. Connect git to your GitHub account with the command:

```
git remote add origin https://github.com/<your_username>/GitTraining.git
```

10. Add some more command to avoid any synchronization issue:

```
git config --global credential.provider generic
```

11. Push the changes to main directory of GitHub

```
git push -u origin main
```

12. Navigate to https://github.com/<your_username>/GitTraining to verify the changes made

Create a GitHub Repository with README file and update the same file

- Steps to Create a Repository from GitHub:
  - Navigate to the link: https://github.com/. Fill the sign-up form and click on “Sign up for Github”.
  - Click on “Start a project” button.
  - Enter any repository name and click on “Create Repository”. User can also give a description to their repository (optional). By default, a GitHub repository is public which means that anyone can view the contents of this repository whereas in a private repository, user can choose who can view the content. Also, private repository is a paid version. User can initialize the repository with a README file. This file contains the description of the file and once user check this box, this will be the first file inside user repository.
  - So now user repository has been successfully created!
- Follow the below steps to perform first commit in GitHub Code:
  - Click on “readme- changes” file which user have created.
  - Click on the “edit” or a pencil icon in the rightmost corner of the file. An editor will open where user can modify the text. Add a commit message which identifies user changes.
  - Click on the "Commit changes" button at the end.
  - User have successfully made their first commit.

Git Configuration and Initilization

1. Create or navigate to a folder and open Git Bash

```
mkdir GitTraining
```

2. Change user terminal to the GitTraining directory with the below command:

```
cd GitTraining
```

3. The git config command is used initially to configure the user.name and user.email. This specifies what email id and username will be used from a local repository. This command sets the author's name and email address respectively to be used with your commits.

```
git config --global user.name “username”
git config --global user.email “useremailaddress”
```

4. Initiate Git with the following command. It creates a .git directory that contains all the Git-related information for user project. After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the repository is initialized, the process of creating other files begins.

```
git init
```

5. Create new file file1.txt. File2.txt and file3.txt in directory and run following command to check status. Status command displays a list the files you’ve changed and those you still need to add or commit.

```
git status
```

Useful command in GIT

7. Enter **ls** in Git Bash to list down all the files. Enter **cat filename** (**cat file1.txt** to get the contents in the file. Run following command to add both files, Add command adds one or more files to staging (index).

git add file1.txt file2.txt file3.txt

8. After staging files, user can commit them into Git. Run following command to commit:

git commit -m “first commit"

User can use -a to commit any files they have added with git add and commit any files user changed since then.

git commit -a

User can use both -a and -m as well

git commit -a -m “first commit“

9. Enter **ls** in Git Bash to list down all the files. Enter **cat filename** to get the contents in the file

10. Modify file1.txt after first commit. Now to check the changes from the last commit, run following command:

git diff

In case user want to have a look at the changes to a particular file, they can run **git diff <file>.**

11. Enter **ls** in Git Bash to list down all the files. Enter **cat filename** to get the contents in the file again

Useful command in GIT

12. Commit the second change changes after modifying file 1.txt

```
git commit –a -m “second commit"
```

13. Verify the history of user project with the following command:

```
git log
```

14. Delete any file from user working directory and stages the deletion:

```
git rm [file]
git rm file 2.txt
```

15. Commit the third change changes after deleting file 2.txt

```
git commit -am “third commit"
```

16. Connect git to your GitHub account with the command:

```
git remote add origin https://github.com/<your_username>/GitTraining.git
```

17. Add some more command to avoid any synchronization issue:

```
git config --global credential.provider generic
```

18. Push the changes to main directory of GitHub

```
git push -u origin master
```

19. Navigate to https://github.com/<your_username>/GitTraining to verify the changes made

Pull Code from GitHub Repository

1. Create or navigate to a folder and open Git Bash

mkdir GitDemo

2. Change user terminal to the GitDemo directory with the below command:

cd GitDemo

3. Initiate Git with the following command.

git init

4. Connect git to your GitHub account with the command:

git remote add origin https://github.com/<your_username>/GitDemo.git

5. The git pull command is used to fetch and download content from a remote repository and immediately update the local repository to match that content.

git pull origin main

6. Create new files file1.txt. file2.txt in directory and run following command to check status. Status command displays a list the files you've changed and those you still need to add or commit.

git status

Create a Branch

7. Enter **ls** in Git Bash to list down all the files. Enter **cat filename** (**cat file1.txt** to get the contents in the file. Run following command to add both files, Add command adds one or more files to staging (index).

git add -A

8. After staging files, user can commit them into Git. Run following command to commit:

git commit -a -m "first commit"

9. Create a new Git branch and checkout in directly with the following command:

git checkout -b <branch_name>

10. Add few more files.

11. Run following command to add both files, Add command adds one or more files to staging (index).

git add -A

12. Commit the second change changes after adding few more files

git commit –a -m "second commit"

13. Checkout to the master branch and Enter **ls** in Git Bash to list down all the files.

git checkout master

Merge the Code

14. Merge the code with new branch

git merge newbranch

15. Enter **ls** in Git Bash to list down all the files in the master branch after merge

16. Checkout to new branch again. Update any files

git checkout newbranch

17. Add command to update the new file to staging (index)

git add -A

18. Commit the changes

git commit -a -m “third commit“

19. Checkout to master branch again.

git checkout master

Push the Code to GitHub after Merging

20. Merge the code with new branch

git merge newbranch

21. Enter **ls** in Git Bash to list down all the files in the master branch after merge. Enter **cat filename (cat file1.txt** to get the contents in the file.

21. Push the code to the newbranch directory of GitHub

git push origin newbranch

22. Push the changes to main directory of GitHub

git push origin main

23. Navigate to **https://github.com/<your_username>/GitTraining** to verify the changes made

Revert any Commit

1. Checkout to new branch again. Create one file (revertchange.txt) add some text.
git checkout newbranch

2. Add command to add the new file to staging (index).
git add revertchange.txt

3. After staging file, user can commit them into Git. Run following command to commit:
git commit -m “add revertchange file“

4. Update the file Add command to add the updated file to staging (index).
git add -A

5. After staging file, user can commit them into Git. Run following command to commit:
git commit -a -m “Update revertchange file“

6. Enter the following command to get the status of commit and get the first 8 didgit of the commit id o revert the last changes
git log

7. Enter the following commnd for revert the last changes in the text file
git checkout firsteightdigitofcommitid revertchange.txt

8. Enter **cat filename** (**cat revertchange.txt** to get the contents in the file.