1.GIT - Distributed Version Control System.

2.Version Control - Having power to go back to Previous Version.To achieve this,we use System called Version Control System.

3.Features : Different Versions, Collaborate with Others Code.

4.Different Version Control Systems : Local Version Control System,Centralized Version Control System,Distributed Version Control System.

5. Distributed Version Control System : GitHub,GitLab,BitBucket.

History of Git:

6.Earlier Days : Archive Files, Sending Patch Files.

7.2002- BitKeeper(send & Merge Code) - Linux.

8.2005 - Git Started - Community Moved from BitKeeper.

9.Mac - XCode(Default Git)

Installation & Commands:

10.git -- version

11.git config

12.git config --global --list - give info about Username,email

13. How to Set username and Email : git config --global user.name “sundhar”

git config –global user.email “[somasundharreddyl@gmail.com](mailto:somasundharreddyl@gmail.com)”

i) To Update Global Config : nano ~/.gitconfig

Git Commands:(In Local Repo)

14.working Directory —> git init —->Staging Area is created to Control over by Git.create git repo in local Repo.

15.git status —-> on Branch master

16. To Rename master to main : Delete git folder and write Command → git init -b main

17. Git status —> on branch main

18.Staging File : Move file to Staging Area from Working Directory: git add filename

19.Unstage File : git rm –cached filename or git restore –staged filename

20. git commit -m”added newline” —> it creates unique 14-Digit Checksum(fd914d5) for every Commit using SHA-1 Alg.

20. To know all Previous Commits : git log

21. Commit file without Staging : git commit -a -m “added new Line”

22. To Know Changes That made before Staging : git diff

23. To Know Changes That made after Staging : git diff –staged

24. By Mistakely You add and commit File in git Repo. You want to Remove it. —> git rm --cached filename.txt and then do Commit.

25. vi filename.txt —-> To edit Content in File.

Move To Remote Repository:

26.BitBucket(Atlassian),GitLab(Private),GitHub(Public)

27.Create Remote Repo —-> +

28. Create Local Repo in Local System.

1. Mkdir git-course or touch filename.txt —-> Create New Folder or File.
2. Cd git-course
3. ls
4. echo “# git-course Demo” >> README.md
5. To Read Content in File : cat filename
6. To get Hidden folders : ls -a
7. git add README.md
8. git commit -m”added readme file”

29.Connect Remote Repo & Local Repo:

1. HTTPS - everytime You need to login to push code into Remote Repo.
2. SSH - Enough to login one Time .

a. Create a gen Key Val: ssh-keygen -o

b.Enter

c.Yes → y

d. Enter (then .ssh folder is generated)

e. cd .ssh

f. ls -a (contains id\_rsa, id\_rsa.pub)

g.cat id\_rsa.pub (it shows generated Key)

h.copy that generated Key in Remote Repository(settings—>SSH and GPG Keys → New SSH Key —>Paste Key. (Now it auto Verifies every time You Push Code into Remote Repo either it is coming from Right Machine).

30.Now Move to Working Repo in Local System : cd git-course

31. Connection made btw Local & Remote : git remote add origin sshLink or https

32.Push your Code into Remote Repo : git push -u origin main

All Commands :

33.pwd (Present Working Directory)

34.rm -rf filename —> Delete File

35.git reset f495ce34526ff — > Changes made in previous Commit are Revert Back.

36.git stash —-> backstage the changes(Store) , without commit.

37.git stash pop -> retrieving the changes from BackStage.

38.git stash clear — > clear the BackStage Changes

39. git remote -v — > Get All URL connected with Folder.

40. git push origin master → commit in Remote.

41.git branch sundhar → Create New Branch

42. git checkout sundhar → move to sundhar Branch . ex: git checkout main

43.git add . & git commit -m”added Msg”

44. git push origin sundhar

//git merge branchName1 —> Merge branch code into main Branch.

Contribution:

44. Fork the Project.

45. git clone https: —--> Copy into Your Local System.

46. git remote add upstream <https://githb> —--> contributor URL(Upstream).

47. Compare & Pull Request

Note ➕

When You are Working on n Features, Create n No of Branches. Because 1 branch Create One Pull Request(Requesting to Merge our code into Contributor Main Code).Don’t push into main branch when you do contribution,create new branch & open New Pull Request.

48. git reset checksum(a43ewqfr5s32ghtr9)

Scenario : You merge and pull the Change in Remote Repository, You want to Undo that Change then Copy Checksum from git log, the one below of suitable commit.

B. git add .

C. git stash

D. git push origin sundhar -f because Remote Repo have Commits,But not Local.

49. Scenario : Update Forked Repo match with main Branch Repo changes.

a.git checkout main

b. git fetch --all --prune //Deleted ones

c. git reset --hard upstream/main

d. git push origin main

or

50. git remote add upstream URL —> Setup the Upstream Remote

git pull upstream main —> Pulling code from contributor main Branch to update in Local

git push origin main —-> Pushing updated Local Code into our Remote Main Branch.

Merge Conflicts:same line changes done by two Persons.Which one to Consider.

51.Squashing Commits : Make two or More commits as single Commit.

→ git rebase -i checksum(4f532sgyrwa76jgt)

Note :

I - insert

Esc - exit Insert Mode

:w → save changes

:q → quit Editor

52. pick,s

53. Sync Main Branch With Your Branch.

1. git checkout main
2. git pull origin main
3. git checkout sundhar
4. Git merge main
5. Git push origin sundhar

54.create an additional commit in your branch. And then remove that commit that has been pushed from the pull request created in previous challenges. The main purpose of this is because there can be times when you might push an unncessary or wrong commit and you way want to remove it from your pull request. This can be done by using the following command: git reset So the task is to identity the commit id of the commit you want to remove, use the git reset command, and remove the commit from the pull request by force pushing into it

1. git log
2. git reset --hard <commit\_id>
3. git push --force origin your-branch-name

—------------------------------------------------------------------------------------------------------------------------------------------

—> Git Rebase to squash the last two commits into one commit:

To squash the last two commits into one commit using Git, you can use the `git rebase` command with the interactive mode. Here are the steps to do this:

1. Open your terminal or command prompt.

2. Make sure you are on the branch where you want to squash the commits. You can use the `git branch` command to see your current branch and switch to the appropriate branch if needed using `git checkout`.

3. Run the following command to start an interactive rebase:

```bash

git rebase -i HEAD~2

```

This command specifies that you want to interactively rebase the last two commits (`HEAD~2`).

4. Your default text editor will open with a list of the commits you're about to rebase. It will look something like this:

```plaintext

pick abc123 Commit message 1

pick def456 Commit message 2

# Rebase 1234567..890abc1 onto 1234567

#

# Commands:

# p, pick = use commit

# r, reword = use commit, but edit the commit message

# e, edit = use commit, but stop for amending

# s, squash = use commit, but meld into previous commit

# f, fixup = like "squash", but discard this commit's log message

# x, exec = run command (the rest of the line) using shell

#

# If you remove a line here THAT COMMIT WILL BE LOST.

# However, if you remove everything, the rebase will be aborted.

#

# Note that empty commits are commented out

```

5. To squash the last two commits into one, change the word `pick` to `squash` (or just `s` for short) for the second commit, like this:

```plaintext

pick abc123 Commit message 1

squash def456 Commit message 2

```

6. Save and close the text editor.

7. A new text editor window will open to allow you to edit the commit message for the squashed commit. You can keep the existing message or modify it as desired.

8. Save and close the text editor.

9. Git will now perform the rebase, squashing the two commits into one. If there are any conflicts during this process, Git will pause and allow you to resolve them.

10. Once the rebase is complete, your branch will have the last two commits squashed into a single commit.

11. You may need to force-push the branch to update the remote repository with your changes:

```bash

git push --force Origin Your\_branchName

```

Remember that force-pushing can be disruptive to collaborators, so use it with caution, especially if you're working in a shared repository.

Git Stash

—------------------

`git stash` command is used to temporarily save changes that you've made in your working directory but don't want to commit just yet. This can be useful in several scenarios:

1. \*\*Switching Branches\*\*: When you are working on one branch and need to switch to another branch but don't want to commit your changes.

2. \*\*Pulling Changes\*\*: When you want to pull changes from a remote repository but can't do so because you have uncommitted changes in your working directory.

3. \*\*Temporary Backups\*\*: When you want to save your current work, for example, to test something quickly and then return to your original state.

Here's how `git stash` works:

1. \*\*Stash Changes\*\*: To stash your changes, you can simply run:

git stash

This command takes all your changes in the working directory (both staged and unstaged) and saves them in a new stash. Git will then revert your working directory to the state of the last commit.

2. \*\*List Stashes\*\*: You can view a list of your stashes using:

git stash list

This will show you a list of stashes with a name and a message.

3. \*\*Apply Stash\*\*: To apply a stash and reapply the changes to your working directory, you can use:

git stash apply stash@{N}

Replace `stash@{N}` with the specific stash you want to apply (you can find the stash name in the list generated by `git stash list`). This will apply the changes but keep the stash intact.

4. \*\*Pop Stash\*\*: To apply a stash and remove it from the list, you can use:

git stash pop stash@{N}

This will apply the changes and remove the stash from the list.

5. \*\*Clear Stash\*\*: To remove a stash without applying it, you can use:

git stash drop stash@{N}

6. \*\*Clear All Stashes\*\*: To remove all stashes, you can use:

git stash clear

Remember that stashes are a handy way to temporarily save your work, but they should not be used as a long-term storage solution. If you want to save your work permanently, it's better to commit your changes with a meaningful commit message. Stashes are meant for short-term storage and convenience.

Github Gists

—----------------------

Certainly! Here's a step-by-step guide to perform the tasks you've mentioned:

1. \*\*Create a Gist on Spring Boot:\*\*

a. Visit the GitHub Gists website (https://gist.github.com/) while logged into your GitHub account.

b. Click on the "New gist" button.

c. In the "Filename including extension" field, give your Gist a descriptive name related to Spring Boot, such as "spring-boot-introduction.md".

d. In the main text area, write your content about Spring Boot. You can provide an introduction, key features, or any other information related to Spring Boot.

e. Optionally, you can add a description for your Gist in the "Description" field.

f. Click the "Create secret gist" button if you want to keep it private. Otherwise, click the "Create public gist" button to make it public.

2. \*\*Create a Gist with Java Code:\*\*

a. Follow the same steps as above to create a new Gist.

b. In the "Filename including extension" field, give your Gist a name related to the Java code snippet, like "java-hello-world.java".

c. In the main text area, write the Java code snippet you want to share. For example, a simple "Hello, World!" program.

d. Optionally, you can add a description for your Gist in the "Description" field.

e. Click the "Create secret gist" button for privacy or the "Create public gist" button for a public Gist.

3. \*\*Create a Markdown File:\*\*

a. In your local development environment, create a Markdown file named `gist-solutions.md`. You can use a text editor like Visual Studio Code or Notepad to create this file.

b. Inside `gist-solutions.md`, you can format your submission and add links to the two Gists you created in the following way:

```markdown

# GitHub Gist Solutions

## Spring Boot Gist

- [Link to Spring Boot Gist](link\_to\_your\_spring\_boot\_gist)

## Java Code Gist

- [Link to Java Code Gist](link\_to\_your\_java\_code\_gist)

```

4. \*\*Push to GitHub:\*\*

a. Save your `gist-solutions.md` file.

b. Commit the file to your local Git repository and push it to your GitHub branch. Assuming you already have a branch named `your\_github\_branch`, you can use these commands:

```bash

git add gist-solutions.md

git commit -m "Add Gist links to gist-solutions.md"

git push origin your\_github\_branch

```

5. \*\*Create a Pull Request:\*\*

a. Visit your GitHub repository's page on the GitHub website.

b. Click the "Pull Requests" tab.

c. Click the "New Pull Request" button.

d. Set the base branch to the branch you want to merge your changes into (e.g., `main` or `master`).

e. Set the compare branch to `your\_github\_branch`.

f. Click the "Create Pull Request" button.

g. Add a title and description to your pull request, if necessary.

h. Click the "Create Pull Request" button again to submit your pull request.

Now, you've successfully created two Gists and a Markdown file that links to them. You've also pushed the changes to your GitHub branch and created a pull request to merge the changes into the main branch of your repository.

Creating a merging conflict

1. \*\*Initialize a Git repository:\*\*

Open your terminal and navigate to the desired directory for your project. Run the following commands to create a new Git repository and set it up.

```bash

mkdir merge-conflict-demo

cd merge-conflict-demo

git init

```

2. \*\*Create a new file and add initial content:\*\*

Create a file named `example.txt` and add some content.

```bash

echo "This is some initial content." > example.txt

git add example.txt

git commit -m "Initial commit with the file"

```

3. \*\*Create branches and make conflicting changes:\*\*

Create two branches, `branch1` and `branch2`, and modify the same line in the file differently in these branches.

```bash

git checkout -b branch1

echo "Change made in branch1." >> example.txt

git add example.txt

git commit -m "Change in branch1"

git checkout master # or branch2

git checkout -b branch2

echo "Change made in branch2." >> example.txt

git add example.txt

git commit -m "Change in branch2"

```

4. \*\*Merge the branches to create a conflict:\*\*

Attempt to merge `branch1` into `master` (or `branch2` into `master`) to intentionally create a merge conflict.

```bash

git checkout master

git merge branch1

```

Git will notify you of a merge conflict. You should see a message like "CONFLICT (content): Merge conflict in example.txt".

Now you have successfully created a merge conflict, and you can proceed to resolve it manually following the steps provided earlier. If you encounter any further issues or have specific questions, feel free to ask.