Git Assignment

**07-03-2025**

**Suneetha.V**

**Basic Git Questions**

1. **What is Git and why is it used?**

**Git is a version control system used to track changes in files and collaborate on software development projects.**

**It helps developers manage changes to code, work together, and maintain project history.**

1. **Explain the difference between Git and GitHub.**

**Git:**

**Git is a version control system that allows you to track changes in your code locally.**

**GitHub:**

**Github is a cloud-based platform that hosts Git repositories**

**online, enabling collaboration and sharing of code.**

1. **How do you install Git on your machine?**

**To install Git:**

* 1. **Download the Git installer from git-scm.com.**
  2. **Run the installer and follow the setup steps (choose default options).**
  3. **Verify the installation by running git --version in your command line**.
  4. **How do you configure your username and email in Git?**

**You can configure your username and email by using the following commands:**

1. **git config --global user.name "Your Name"**
2. **git config --global user.email "**[**youremail@example.com**](mailto:youremail@example.com)**" ex:**
   1. **git config --global user.name "Suneetha"**
   2. **git config --global user.email "**[**suneethavemula@puropalecreations.com**](mailto:suneethavemula@puropalecreations.com)**"**
   3. **What is a repository in Git?**

**A repository in Git is a storage location where all files, commit history, and branches of a project are kept. It can be local (on your machine) or remote (on platforms like GitHub).**

* 1. **How do you create a new Git repository?**

**To create a new Git repository:**

1. **Open the terminal/command line.**
2. **Navigate to your project folder.**
3. **Run the command: git init**

**This will create a new Git repository in that directory.**

* 1. **How do you clone a repository from GitHub?**

**To clone a repository from GitHub:**

1. **Go to the repository's GitHub page.**
2. **Copy the repository's URL (HTTPS or SSH).**
3. **In the terminal, run:**

**git clone <repository\_URL>**

**This will create a local copy of the repository.**

* 1. **What is the purpose of the .gitignore file?**

**The .gitignore file tells Git which files or directories to ignore (not track) in a repository. This is used to exclude temporary files, build files, or sensitive data like passwords**.

* 1. **How do you check the status of your working directory in Git?**

**To check the status of your working directory, use the following command:**

**git status**

**This will show which files are modified, added, or not yet staged for commit.**

* 1. **How do you add files to the staging area in Git? To add files to the staging area, use the command: git add <filename>**

**Or, to add all files:**

git add .

**This prepares the files to be committed. Intermediate Git Questions**

* 1. **Explain the concept of commits in Git.**

**A commit in Git is a snapshot of your project at a particular point in time.**

**Each commit records changes to the files in the repository and includes a commit message describing what was changed.**

**Commits help you track the history of the project, allowing you to revert to previous versions or examine changes made over time.**

* 1. **How do you create a new commit in Git?**

To create a new commit, follow these steps:

Stage the changes by adding the modified files to the staging area using:

git add <filename> # Or use `git add .` to add all changes

Commit the changes with a message that describes the changes:

git commit -m "Your commit message"

This will create a new commit that includes the staged changes and the message.

* 1. **What is the purpose of the git log command?**

The git log command is used to display the commit history of a Git repository.

It shows the details of each commit, such as the commit hash, author, date, and the commit message. This is helpful for tracking changes over time and reviewing project history.

* 1. **How do you view the history of commits in a repository?**

You can view the history of commits in a repository by using the following command:

git log

This will display a list of commits in reverse chronological order (most recent first). You can use various options (e.g., git log --oneline) to modify the output format.

* 1. **How do you view the changes made in a commit?**

To view the changes made in a specific commit, use the git show command followed by the commit hash:

git show <commit-hash>

This will display the changes (diff) made in that commit, including added or deleted lines in files.

* 1. **What is branching in Git and why is it useful?**

Branching in Git allows you to create separate lines of development within a repository.

Each branch can represent a different feature or bug fix, which helps in managing multiple tasks simultaneously

without affecting the main codebase (usually the main or master branch).

Branching is useful because it lets developers work on isolated changes without interfering with others, making collaboration easier and safer.

* 1. **How do you create a new branch in Git?**

To create a new branch in Git, use the following command:

git branch <branch-name>

This creates the new branch locally. To switch to the new branch after creating it, use:

git checkout <branch-name>

Alternatively, you can create and switch to a new branch in one step with:

git checkout -b <branch-name>

* 1. **How do you switch between branches in Git?**

To switch between branches in Git, use the git checkout command:

git checkout <branch-name>

This changes your working directory to the specified branch. From Git 2.23 onward, you can also use the git switch command:

git switch <branch-name>

* 1. **What is the difference between git merge and git rebase?**

git merge: Merges the changes from one branch into another. This creates a new commit that combines the histories of both branches, preserving the individual branch histories. It may result in a "merge commit."

git rebase: Re-applies commits from one branch on top

of another branch. This rewrites the commit history by moving or "replaying" changes, resulting in a linear history without merge commits.

Use merge if you want to preserve the branch structure. Use rebase if you prefer a cleaner, linear history.

* 1. **How do you resolve merge conflicts in Git? Merge conflicts occur when Git cannot automatically merge changes from two branches. To resolve them:**

Run git merge (or git rebase) to merge the branches. Git will stop and mark the conflicting files.

Open the conflicted files to see sections marked with

<<<<<<<, =======, and >>>>>>>. These indicate the conflicting changes.

Manually edit the file to resolve the conflicts, removing

the conflict markers.

After resolving conflicts, add the resolved files to the staging area:

git add <resolved-file>

Finally, commit the changes (if merging) with: git commit

Or continue the rebase with:

git rebase --continue

Git Exercises

* 1. **Create a new Git repository and configure your username and email.**

Step 1: Initialize a new Git repository

To create a new Git repository, follow these steps:

1. **Open your terminal/command prompt.**
2. **Navigate to the directory where you want to create your project:**

cd /path/to/your/directory Initialize a new Git repository:

git init

Step 2: Configure your username and email

After initializing the repository, you need to configure your username and email (this will be used for your commits):

git config --global user.name "Your Name"

git config --global user.email "[youremail@example.com](mailto:youremail@example.com)" EX: git config --global user.name "Suneetha"

git config --global user.email "[suneethavemula@puropalecreations.com](mailto:suneethavemula@puropalecreations.com)"

* 1. **Create a file, add some content to it, and commit the changes.**

Step 1: Create a file and add some content

Create a new file (for example, example.txt) and add

some content to it:

echo "This is my first commit!" > example.txt Step 2: Add the file to the staging area

After creating the file, add it to Git's staging area: git add example.txt

Step 3: Commit the changes

Now, commit the changes with a descriptive message: git commit -m "Added example.txt with initial content"

* 1. **Create a .gitignore file and add rules to ignore specific files and directories.**

Step 1: Create a .gitignore file

In the root of your Git repository, create a .gitignore file. You can create it using a text editor or directly from the terminal:

touch .gitignore

Step 2: Add rules to .gitignore

Open the .gitignore file and add rules for the files or directories you want to ignore. For example:

# Ignore all .log files

\*.log

# Ignore all .tmp files

\*.tmp

# Ignore the node\_modules directory node\_modules/

# Ignore all .DS\_Store files (macOS)

.DS\_Store

Save the .gitignore file.

* 1. **Clone an existing repository from GitHub and make some changes.**

Step 1: Clone the repository from GitHub

To clone an existing repository, you need the URL of the repository. Go to the repository's GitHub page and copy the URL.

Then, run the following command in your terminal to clone it:

git clone https://github.com/username/repository- name.git

Replace https://github.com/username/repository- name.git with the actual repository URL.

Step 2: Make some changes

After cloning, navigate into the cloned repository's directory:

cd repository-name

Make some changes to a file or create a new one. For example, you can add a new file:

echo "Some changes made" > changes.txt Step 3: Add, commit, and push changes

1. **Add the changes to the staging area: git add changes.txt**
2. **Commit the changes:**

git commit -m "Added changes.txt with some content"

1. **Push the changes to the GitHub repository:**

git push origin main

* 1. **Create a new branch, make some changes, and switch back to the main branch.**

Step 1: Create a new branch

To create a new branch, use the following command: git branch new-branch

This creates a new branch named new-branch but does not switch to it. To switch to the new branch, use:

git checkout new-branch

Alternatively, you can use this single command to create and switch to the new branch:

git checkout -b new-branch

Step 2: Make changes in the new branch

Once you’re on the new branch, make some changes (e.g., create or modify a file):

echo "Changes on new branch" > new-file.txt

Add and commit the changes:

git add new-file.txt

git commit -m "Created new-file.txt on new-branch" Step 3: Switch back to the main branch

After committing your changes in the new branch, you can switch back to the main branch:

git checkout main

Alternatively, if you're using Git 2.23 or later, you can use git switch:

git switch main

Now you’re back on the main branch. If you want to merge the changes from new-branch into main, you can do so by running:

git merge new-branch