EXP – 2: Demonstrate working with Git Shell commands Consider python project requirements

Examples of Python Project Requirements:

1. **Prime checker** # commit as Version – 1
2. **Even Number Generator** # commit as Version – 2
3. **Generate factorial** # commit as Version – 3
4. **Generate a table** # commit as Version – 4
5. **Generate a list of vowels** from a text # commit as Version – 5

To demonstrate working with Git Shell commands in Python using Visual Studio Code (VSCode), follow these steps:

Step 1: Install Git and VSCode

1. **Install Git:**
   - Download and install Git from git-scm.com.
   - During installation, make sure to select the option to add Git to your system PATH.
2. **Install Visual Studio Code:**
   - Download and install VSCode from code.visualstudio.com.

Step 2: Set Up a Git Repository in VSCode

1. **Open VSCode:**
   - Launch Visual Studio Code.
2. **Open the Terminal:**
   - Open the terminal in VSCode by clicking on Terminal -> New Terminal or using the shortcut Ctrl + ` .
3. **Navigate to your project directory:**
   - Use the terminal to navigate to the directory where your project files are located. For example:

```
cd path/to/your/project
```

4. **Initialize a Git repository:**
   - If your project is not already a Git repository, initialize it by running:

```
git init
```

Step 3: Working with Git Commands

Here's a demonstration of basic Git commands you might use in the VSCode terminal:

1. **Check the status of the repository:**

```
git status
```

   - This command will show you the current status of your working directory and staging area.

2. **Add files to the staging area:**

```
git add .
```

   - This command adds all changes in your working directory to the staging area.

3. **Commit changes:**

```
git commit -m "Initial commit"
```

   - This command commits the staged changes with a message.

Step 4: Integrate Git with Python Code in VSCode

1. **Create a Python script:**
   - In VSCode, create a new file named example.py.
   - Write a simple Python script:

```
print("Hello, Git and Python!")
```

2. **Run the Python script:**
   - Use the VSCode terminal or the integrated Run command to execute your Python script.

```
python example.py
```

3. **Track and commit the Python file:**
   - Add the new Python file to the staging area:

```
git add example.py
```

   - Commit the change:

```
git commit -m "Add example Python script"
```

Step 5: Use VSCode Git Integration Features

VSCode also provides an integrated Git experience:

1. **Source Control Panel:**
   - Click on the Source Control icon in the Activity Bar on the side of VSCode.
   - You can see all changes, add them to staging, commit them, and push/pull changes directly from this panel.
2. **GitLens Extension (Optional):**
   - Install the GitLens extension from the VSCode marketplace to get more advanced Git features, such as visualizing commit history, viewing file history, and more.

By following these steps, you can effectively use Git commands within the VSCode terminal while working on your Python projects.

Summary of Git Commands:

Certainly! Here's a list of common Git commands along with their purposes:

1. **git init** - Initializes a new Git repository in the current directory.
2. **git clone <repository>** - Creates a copy of an existing Git repository from a remote source.
3. **git add <file>** - Stages changes made to files for the next commit. You can use git add . to stage all changed files.

4. **git commit -m "<message>"** - Commits the staged changes with a descriptive message.
5. **git status** - Displays the current state of the working directory and the staging area, including untracked, modified, and staged files.
6. **git log** - Shows a log of commits made in the repository, including their hash, author, date, and message.
7. **git diff** - Shows the differences between files or commits. For example, git diff shows changes in the working directory compared to the staging area.
8. **git branch** - Lists all branches in the repository. You can use git branch <name> to create a new branch.
9. **git checkout <branch>** - Switches to the specified branch. You can also use it to restore files from a previous commit or branch.
10. **git merge <branch>** - Merges changes from the specified branch into the current branch.
11. **git pull** - Fetches and merges changes from the remote repository into the current branch.
12. **git push** - Pushes committed changes from the local repository to the remote repository.
13. **git remote** - Manages remote repository connections. git remote -v shows the URLs of the remote repositories.
14. **git fetch** - Retrieves updates from a remote repository but does not merge them into the current branch.
15. **git reset** - Undoes changes by resetting the index or the working directory to a previous commit. It can be used with options like --soft, --mixed, or --hard.
16. **git revert <commit>** - Creates a new commit that undoes the changes made in a specified commit.
17. **git stash** - Temporarily saves changes that are not ready to be committed, allowing you to switch branches or work on other tasks.

18. **git tag** - Creates a reference to a specific commit. Tags are often used to mark release points.

To set your account's default identity

Omit --global to set the identity only in this repository.

```
git config --global user.email "you@example.com"

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