* **GEN AI**
* [**https://github.com/microsoft/generative-ai-for-beginners**](https://github.com/microsoft/generative-ai-for-beginners)
* [**https://www.cloudskillsboost.google/paths/17**](https://www.cloudskillsboost.google/paths/17)
* [**https://campus.datacamp.com/courses/understanding-artificial-intelligence/what-is-artificial-intelligence-ai?ex=5**](https://campus.datacamp.com/courses/understanding-artificial-intelligence/what-is-artificial-intelligence-ai?ex=5)
* **https://github.com/suneelpatel/Statistics-for-Data-Science-using-Python/blob/master/Bayes%20Theorem/Bayes\_Theorem.py**

Git and GitHub

* <https://github.com>
* <https://git-scm.com>
* <https://git-scm.com/book/en/v2/Getting-Started-What-is-Git>
* <https://ourcodingclub.github.io/tutorials/git/>

<https://docs.github.com/en/education/manage-coursework-with-github-classroom/teach-with-github-classroom/create-an-individual-assignment>

<https://github.blog/changelog/2024-01-22-migrating-github-classroom-assignment-repository-creation-from-create-from-template-to-forks-public-beta/>

<https://www.nobledesktop.com/outlines/git-going-an-intro-to-git-and-github-syllabus.pdf>

Section 1 Download & Install Git on Mac & Windows Git Setup: Your Name & Email Create a New Local Git Repository (Initialize Repository) Stage & Commit Files Section 2 GitHub: Push to a Remote Repository GitHub: Pull From a Remote Repository GitHub: Clone (Download) a Remote Repository

**Certainly! Here are some basic Git commands commonly used in production environments:**

1. \*\*git init\*\*: Initialize a new Git repository in the current directory.

2. \*\*git clone <repository>\*\*: Clone a remote repository onto your local machine.

3. \*\*git add <file>\*\*: Add changes in a file to the staging area.

4. \*\*git commit -m "Message"\*\*: Commit changes from the staging area to the repository with a descriptive message.

5. \*\*git status\*\*: Check the status of files in the working directory, staging area, and repository.

6. \*\*git pull\*\*: Fetch changes from the remote repository and merge them into the current branch.

7. \*\*git push\*\*: Upload local repository changes to a remote repository.

8. \*\*git branch\*\*: List all local branches in the repository.

9. \*\*git checkout -b <branch>\*\*: Create a new branch and switch to it.

10. \*\*git merge <branch>\*\*: Merge changes from another branch into the current branch.

11. \*\*git remote -v\*\*: List all remote repositories and their URLs.

12. \*\*git log\*\*: Display a log of all commits in the repository.

13. \*\*git diff\*\*: Show changes between commits, commit and working tree, etc.

14. \*\*git reset\*\*: Reset the current HEAD to a specified state.

15. \*\*git stash\*\*: Temporarily shelve changes that are not ready to be committed.

These commands cover the basic workflow for collaborating on a project, managing branches, committing changes, and interacting with remote repositories, making them essential for production environments.

**Section 3**

**Section 3 How to Handle Merge Conflicts**

**View a List of Commits & Undo Changes**

**Branches: Create, Switch, Push, Merge, & Delete**

\*\*Code Example: Handling Merge Conflicts\*\*

Suppose you're working on a collaborative project with other developers using Git and GitHub. Here's how you can handle merge conflicts:

1. Pull Changes from the Remote Repository:

```bash

git pull origin main

```

2. Resolve Merge Conflicts:

Open the conflicted file(s) in your code editor and manually resolve the conflicts. Git markers (`<<<<<<<`, `=======`, `>>>>>>>`) indicate the conflicting sections. Edit the file to keep the desired changes and remove the markers.

<https://www.simplilearn.com/tutorials/git-tutorial/merge-conflicts-in-git>

3. Add the Resolved Changes:

```bash

git add <conflicted-file>

```

4. Commit the Changes:

```bash

git commit -m "Resolved merge conflicts"

```

5. Push the Changes to the Remote Repository:

```bash

git push origin main

```

\*\*Demo:\*\*

Watch a demo of handling merge conflicts in a Git repository [here](https://www.youtube.com/watch?v=JtIX3HJKwfo).

\*\*Use Case:\*\*

Imagine you're collaborating with a team of developers on a project hosted on GitHub. Two team members make conflicting changes to the same file and push their changes to the remote repository. When you try to pull the latest changes, Git detects a merge conflict. By following the steps above, you can efficiently resolve the conflict, ensuring the project's codebase remains consistent and functional.

\*\*Code Example: Viewing a List of Commits & Undoing Changes\*\*

To view a list of commits and undo changes in Git:

1. View Commit History:

```bash

git log

```

2. Identify the Commit to Undo:

Note the commit hash or index of the commit you want to revert.

3. Undo Changes:

```bash

git revert <commit-hash>

```

4. Resolve any Conflicts and Commit the Changes:

If there are conflicts during the revert process, resolve them as described earlier.

5. Push the Changes to the Remote Repository:

```bash

git push origin main

```

\*\*Demo:\*\*

Watch a demo of viewing commit history and undoing changes in Git [here](https://www.youtube.com/watch?v=RF9wAqAVAEQ).

\*\*Use Case:\*\*

Suppose you accidentally introduced a bug into your project after making several commits. By viewing the commit history and identifying the commit that introduced the bug, you can use the `git revert` command to undo the changes introduced by that commit while preserving the project's history and avoiding disruptions to your team's workflow.

\*\*Code Example: Branches Operations\*\*

Git branches are essential for managing project development. Here's how to perform common branch operations:

1. Create a New Branch:

```bash

git checkout -b new-branch

```

2. Switch to an Existing Branch:

```bash

git checkout existing-branch

```

3. Push a Branch to the Remote Repository:

```bash

git push origin branch-name

```

4. Merge a Branch into the Main Branch:

```bash

git checkout main

git merge branch-name

```

5. Delete a Branch:

```bash

git branch -d branch-name

```

\*\*Demo:\*\*

Watch a demo of creating, switching, pushing, merging, and deleting branches in Git [here](https://www.youtube.com/watch?v=6l1S5DiOIYY).

\*\*Use Case:\*\*

When working on a new feature or fixing a bug, you can create a new branch to isolate your changes from the main codebase. Once the changes are complete and tested, you can merge the branch into the main branch. Deleting unnecessary branches helps keep the repository clean and organized.

## **Git Assignment # 1**

**Write a 1 Post on Debug.School which would Answer following question…**

* What is Git?
* How git works? Architecture of git.
* Exaplain Why Git is Distributed?
* Explain Git Workflow with image
* List of Top 10 Git commands
* What is Version Control?
* What is a commit?
* What is a merge?
* What is a merge conflict?
* What best practices are you familiar with regarding version control?

## **Git Assignment # 2**

**Write a 1 Post on Debug.School which would Answer following question…**

1. What is Git and why is it important in software development?
2. Explain the difference between Git and GitHub.
3. How does Git handle version control?
4. Discuss the three main states of a Git file: committed, modified, and staged.
5. How can you create a new Git repository and clone it to your local machine?
6. What is the difference between a Git branch and a tag?
7. How do you merge changes from one branch to another in Git?
8. Explain the purpose of the .gitignore file in Git.
9. Discuss the Git workflow for a team of developers.
10. How can you resolve a Git conflict when merging branches?
11. What is the significance of the Git HEAD pointer?
12. Discuss the difference between a Git pull and a Git fetch.
13. Explain the difference between Git rebasing and Git merging.
14. How can you recover a deleted branch in Git?

## **Git Labs & Excercise**

[Git Lab Exercise & Assignment: Basic local workflow: Part – 1](https://www.devopsschool.com/blog/git-lab-exercise-assignment-basic-local-workflow-part-1/)

[Git Lab Exercise & Assignment: Remote workflow: Part – 2](https://www.devopsschool.com/blog/git-lab-exercise-assignment-remote-workflow-part-2/)

[Git Lab Exercise & Assignment: Working with Local Branch: Part – 3](https://www.devopsschool.com/blog/git-lab-exercise-assignment-working-with-local-branch-part-3/)

[Git Lab Exercise & Assignment: Branching & Merging & Conflicts: Part – 4](https://www.devopsschool.com/blog/git-lab-exercise-assignment-branching-merging-conflicts-part-4/)

[Git Lab Exercise & Assignment: Changing History with Amend: Part – 5](https://www.devopsschool.com/blog/git-lab-exercise-assignment-changing-history-with-amend-part-5/)

[Git Lab Exercise & Assignment: Local Branches with Rebasing: Part – 6](https://www.devopsschool.com/blog/git-lab-exercise-assignment-local-branches-with-rebasing-part-6/)

[Git Lab Exercise & Assignment: Git remote branching and github: Part – 7](https://www.devopsschool.com/blog/git-lab-exercise-assignment-git-remote-branching-and-github-part-7/)

[Git Lab Exercise & Assignment: Git Branching and Merging: Part – 8](https://www.devopsschool.com/blog/git-lab-exercise-assignment-git-branching-and-merging-part-8/)

[Git Lab Exercise & Assignment: Git Diff and Undoing: Part – 9](https://www.devopsschool.com/blog/git-lab-exercise-assignment-git-diff-and-undoing-part-9/)

[Git Lab Exercise & Assignment & Project: Part – 10](https://www.devopsschool.com/blog/git-lab-exercise-assignment-project-part-10/)

[Git Lab Exercise & Assignment: Git Stash: Part – 11](https://www.devopsschool.com/blog/git-lab-exercise-assignment-git-stash-part-11/)

[Git Lab Exercise & Assignment: Git cherry-pick: Part – 12](https://www.devopsschool.com/blog/git-lab-exercise-assignment-git-cherry-pick-part-12/)