**Git branches**

In Git, a branch is simply a lightweight movable pointer to a commit. The default branch is called **master** in most repositories, but it can be renamed to **main** or another preferred name. Each branch represents an independent line of development in your project.

**What is a Git Branch?**

A Git branch is a pointer to a specific commit within the repository. It allows you to work on multiple versions of your project simultaneously. Each branch is an independent line of development.

**Why Use Branches?**

* **Isolation**: Branches allow you to work on new features, bug fixes, or experiments without affecting the main codebase.
* **Collaboration**: Different team members can work on different features simultaneously by creating separate branches.
* **Experimentation**: Branches are great for trying out new ideas without disrupting the main codebase.
* **Versioning**: You can use branches to maintain different versions of your project simultaneously.

**Git commands for branches**

**1) Create a Branch**: You can create a new branch with the **git branch** command:

**git branch new\_branch\_name**

**2) Switch Branches**: Use the **git checkout** command to switch to a different branch:

**git checkout branch\_name**

**3) Create and Switch**: You can create a new branch and switch to it in one command using **-b** option:

**git checkout -b new\_branch\_name**

**4) List Branches**: To list all branches in your repository, you can use:

**git branch**

**5) Merge Branches**: Once you've finished work on a branch, you can merge it back into the main branch (e.g., **master**) using:

**git checkout master**

**git merge branch\_name**

**6) Delete Branches**: After merging a branch, you can delete it with:

**git branch -D branch\_name**

**Deleting a Remote Branch:**

git push origin --delete branchname

**Renaming a Branch:**

git branch -m old-branch-name new-branch-name

**Handling Merge Conflicts:** If there are conflicts during a merge, Git will pause the merge and allow you to resolve the conflicts manually. After resolving, you need to add the resolved files and complete the merge.

**Best Practices for Branching**

1. **Use Descriptive Names:** Choose meaningful branch names that reflect the purpose of the branch, such as feature-login, bugfix-issue123, or release-v1.2.
2. **Keep Branches Short-Lived:** Regularly merge changes back into the main branch to avoid large merge conflicts.
3. **Regularly Update Branches:** Frequently pull changes from the main branch to keep your branch up-to-date and minimize conflicts.
4. **Use Feature Branches:** Create a new branch for each new feature, bug fix, or experiment.
5. **Clean Up Stale Branches:** Regularly delete branches that are no longer needed to keep the repository clean.

**Git Conflicts**

**What is a Git Conflict?**

A Git conflict occurs when changes from different branches or contributors cannot be automatically merged.

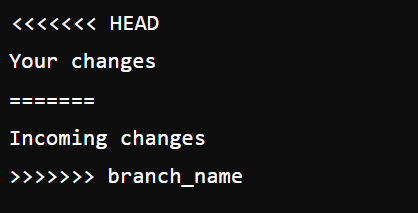
Conflicts typically arise when two or more branches have modified the same lines of a file or when one branch has deleted a file that another branch has modified.

**Common Causes of Git Conflicts**

1. **Concurrent Changes**: Multiple developers making changes to the same lines in a file.
2. **Deletions and Modifications**: One branch deletes a file while another modifies it.
3. **Branching and Merging**: Frequent branching and merging without synchronization.

**Identifying Conflicts**

* Git automatically flags conflicts during a merge or rebase.
* Conflict markers are inserted into files to highlight the differing changes.
* Example of conflict markers:



*Resolving Conflicts*

1. **Manual Resolution**:
   * Open the conflicted file.
   * Decide which changes to keep, modify, or combine.
   * Remove conflict markers.
2. **Use Git Tools**:
   * Use GUI tools like GitKraken, Sourcetree, or VSCode's built-in merge conflict resolver.
   * These tools provide a visual interface to resolve conflicts.
3. **Commit Resolution**:
   * After resolving conflicts, stage the resolved files:

git add <resolved\_file>

* + Commit the resolution:

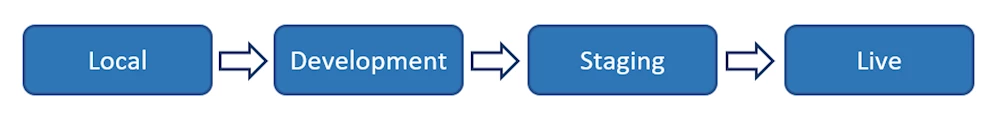
git commit

**where to resolve issues by researching**

* stack overflow
* medium
* docs
* chatgpt

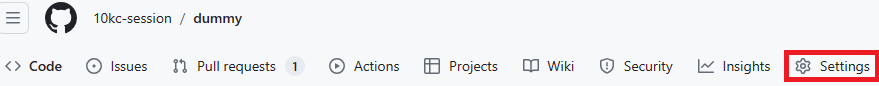
**DEPLOYMENT**

* Deployment in software and web development means pushing changes or updates from one deployment environment to another.
* In simple terms development stage to production stage
* After Deployment of any website, we can access from any where these are called **live websites / webpages**

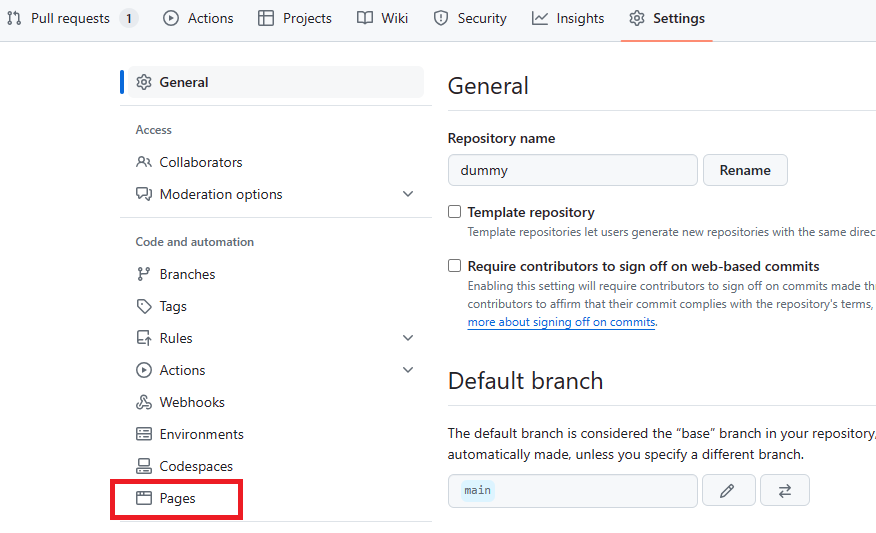
****

**STEPS TO DEPLOY WEBPAGES INTO LIVE SERVER / GITHUB**

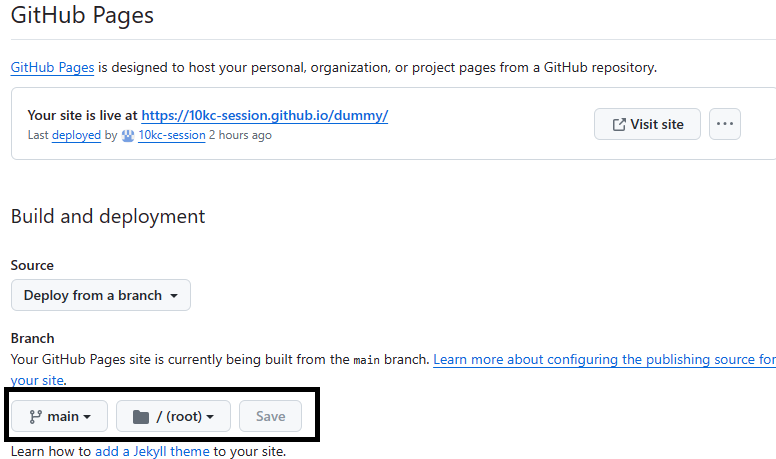
After Adding Code to Repository, go-to settings of repository

****

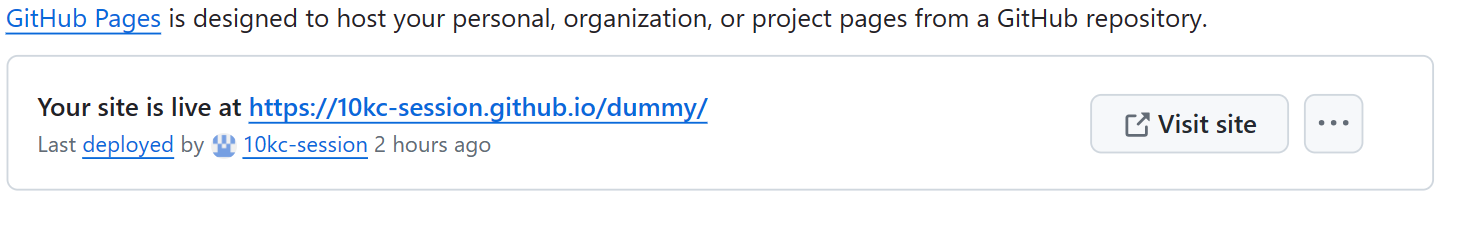
After clicking on settings click on pages which is shown below



After opening pages, you will find below page (GitHub Pages), Now Click on branch section and change branch if needed and simply click on **save**



After Clicking on save after few minutes refresh the page and you will find the link of your website



Now Click On The Link And Your Website is Live..