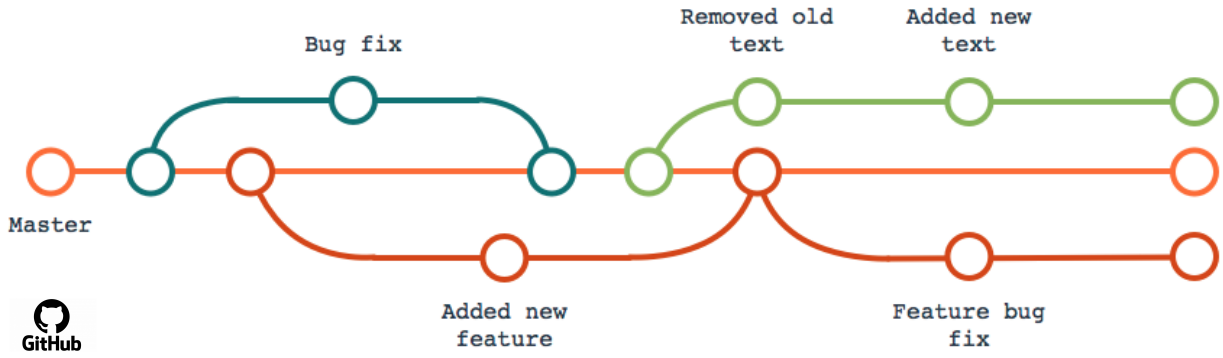
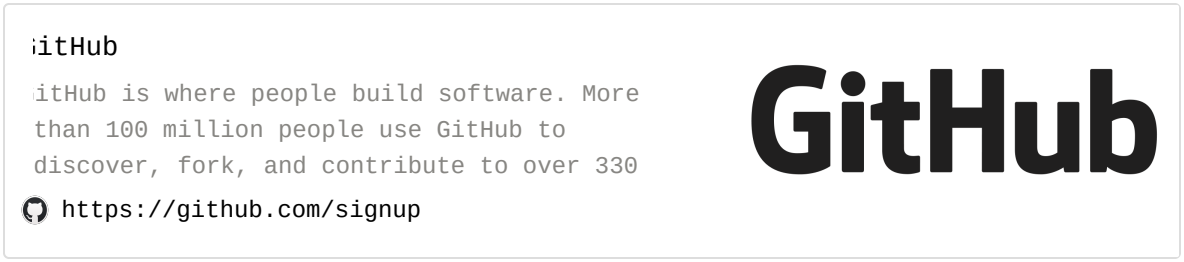


Git | Tutorial

Setup Github

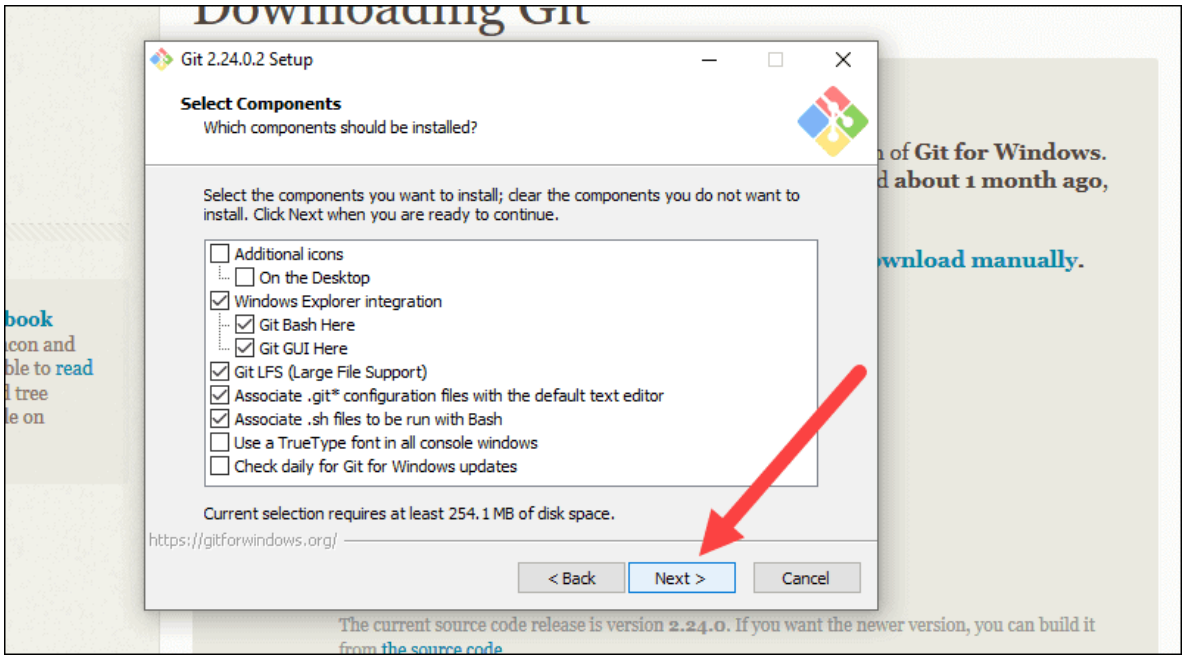
1. Create a Github account first : **https://github.com/signup**



   Make sure to note down your username and password somewhere.

2. **Install Gitbash** : Visit the official Git website: **https://git-scm.com/** and download the latest version
3. During installation just tick on *"Additional icons + on desktop" and also make sure windows explorer integration and its respective sub-options i.e. "Git Bash Here" and "Git Gui Here" are ticked* after that dont modify any other option and

just click next until an install option is shown to you and then just click install.



4. Open any folder where you want to start creating a repository and just right click and choose **“open gitbash here”** in case if you are using windows 11 you will have to choose **more options** and then choose the **“open gitbash here”**

5. Use the below command

   WARNING :

   type these commands and **don’t copy-paste it** because sometimes it happens that you may accidently execute the command by directly pasting it in the terminal of git bash and it will not wait for you to add username or email there.

```
git config --global user.name Your_Username_Here_As_Per_Github

 git config --global user.email Your_Email_Id_Of_Github
```

What is Git | What is Github and Gitlab?

GIT

Git is a distributed version control system that allows you to track changes to your code, collaborate with others, and easily manage different versions of your project. It keeps a complete history of all the changes made to your files, allowing you to revert to previous versions or merge changes from multiple contributors.

GITHUB

GitHub is a web-based platform centered on Git. It hosts Git repositories, enabling easy sharing, collaboration, and contribution. With features like issue tracking, pull requests, and project management tools, it serves as a central hub for developers to showcase their work and collaborate effectively.

GITLAB

GitLab is a web-based platform for hosting Git repositories and facilitating collaboration in software development. Unlike GitHub, GitLab offers both cloud-based and self-hosted versions (GitLab CE/EE), granting organizations greater control over their repositories and data.

Why we need Git ?

Version Control: Git enables tracking and managing code changes over time, providing a history of every modification made to files. It allows easy reversion to previous versions and comparison of differences between versions.

Collaboration: Git promotes seamless collaboration among multiple developers, enabling simultaneous work on projects without conflicts. It provides a structured approach to managing code contributions from individuals or teams.

Branching and Merging: With Git, create isolated branches for new features, bug fixes, or experiments without impacting the main codebase. Test changes independently and merge them back when ready.

Code Integrity: Git ensures code integrity by uniquely identifying each commit with a hash. It enables recovery from corruption or errors through distributed repositories.

Time Travel: Git serves as a code time machine, allowing easy navigation to any previous commit, inspection of code states, and creation of new branches. It aids issue troubleshooting, understanding code changes, and reverting to stable states if needed.

Experimentation and Safe Testing: Git encourages safe experimentation by providing a separate environment for trying new ideas. Create branches, test changes, and discard them if experiments fail. It enables quick iteration and maintains a stable main branch.

Open Source Contribution: Git facilitates global collaboration in the open-source community, enabling project contributions, pull requests, and software development for the benefit of all.

Code Review and Quality Control: Git streamlines code review processes by allowing team members to leave comments and suggest

improvements. It ensures code quality and alignment with project requirements and standards.

Continuous Integration and Deployment (CI/CD): Git seamlessly integrates with CI/CD pipelines, automating testing, building, and deployment. It ensures controlled and efficient integration, testing, and deployment of code changes.

What is a Repository ?

A repository, often referred to as a "repo," is a central location where Git stores all the files and their version history.

In Git, there are two types of repositories: **local and remote.**

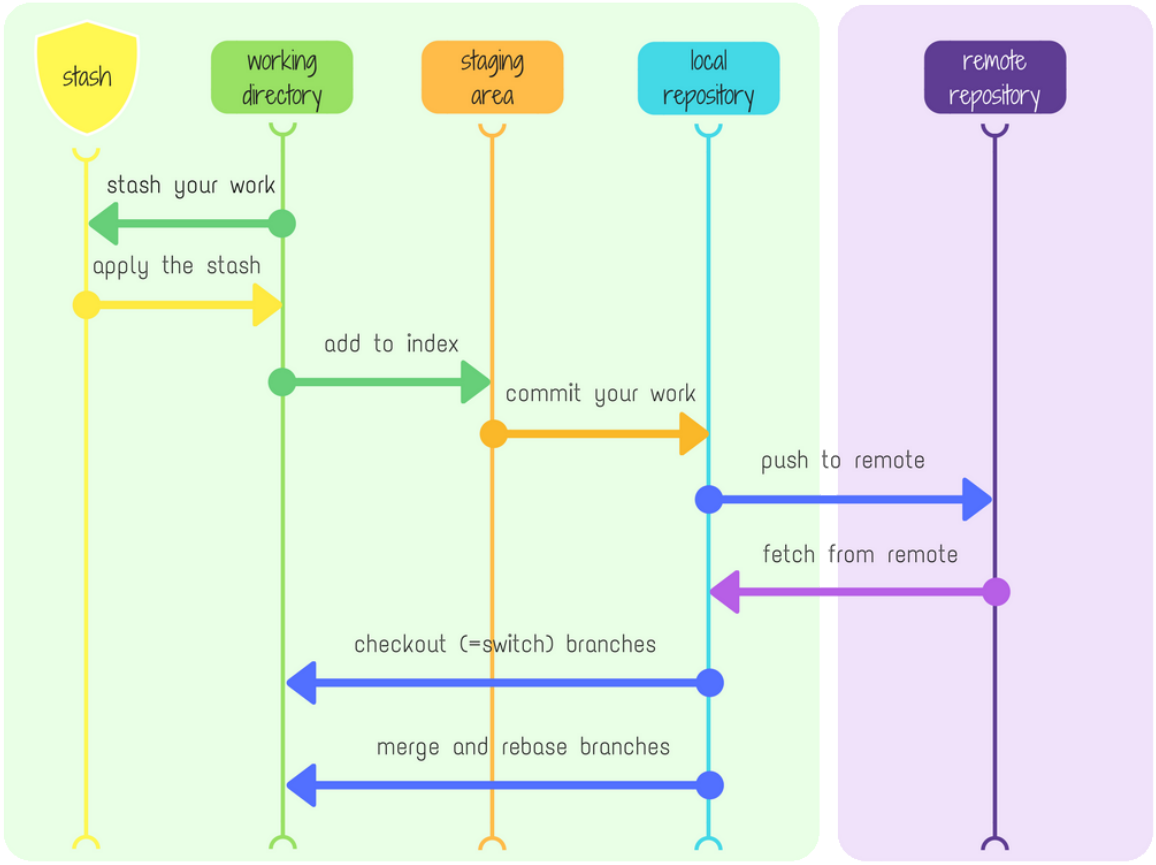
1. **Local repositories** are copies on your machine, containing the entire version history. You can make changes, commit them, and perform Git operations within them.
2. **Remote repositories**, hosted on servers like GitHub and GitLab, serve as centralized locations for collaboration. They can be accessed by cloning or forking. Fetching and pushing changes synchronize the codebase between local and remote repositories.
3. In summary, **local repositories reside on your machine**, **while A remote repository is a version of the Git repository hosted on a different server or platform**.

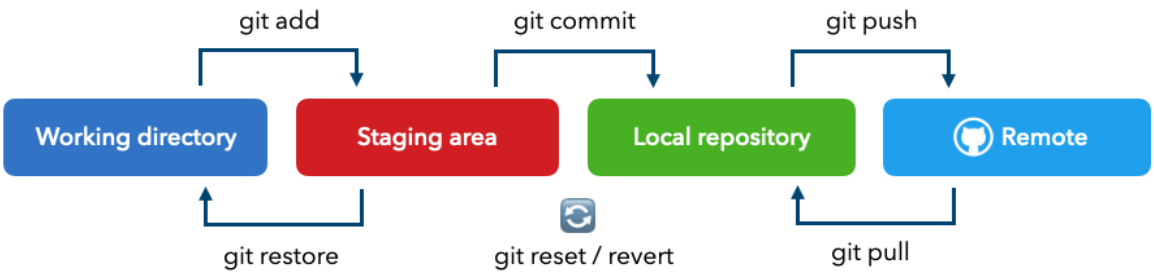
What are some operational areas in Git ?

1. **Stash Area:**
    - The stash area is a temporary storage in Git used to save changes that are not ready to be committed.
    - It allows you to store modifications that are in progress, allowing you to switch branches or perform other operations without committing unfinished work.
    - Stashing is useful when you need to temporarily set aside changes and switch to a different task or branch.
2. **Working Directory:**
    - The working directory, also known as the working tree, is the directory on your local machine where you have your Git repository cloned.
    - It contains the actual files of your project that you can edit and modify.
    - When you make changes to files in the working directory, Git tracks these modifications as differences from the last commit.
3. **Staging Area (Index):**
    - The staging area, also referred to as the index, is an intermediate area between the working directory and the local repository.
    - It acts as a holding area for changes that you want to include in the next commit.
    - Before creating a commit, you need to explicitly add files or changes to the staging area using the `git add` command.
4. **Local Repository:**

- The local repository is a complete copy of the Git repository stored on your local machine.
- It includes the full history of commits, branches, tags, and other Git objects.
- The local repository allows you to perform Git operations and version control on your local machine without requiring a network connection.

5. **Remote Repository:**
    - The remote repository is a Git repository located on a remote server or hosting platform, such as GitHub or GitLab.
    - It serves as a centralized location where multiple developers can collaborate and share their changes.
    - Remote repositories provide a way to synchronize your local repository with the changes made by others and vice versa.
    - You can push your local commits to the remote repository and pull changes from the remote repository to update your local repository.





Git Commands :

Caution : Please don’t copy paste command becasue sometimes it automatically executes the command and you wont be able to make changes to it

1. **Clone a Repository**:
    - To get a local copy of your repository, use the `git clone` command:Replace `<repository-url>` with the URL of your GitHub repository. This will create a new directory with the same name as your repository.

```
git clone <repository-url>
```

2. **Configure Git:**
    - **Set your username:**

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

    - **Set your email address:**

```
git config --global user.email "your-email@gmail.com"
```

    - These configurations are important to associate your commits with your GitHub account.
3. **Stage Files:**
    - To stage selected multiple files, use the `git add` command followed by the file names:

```
git add file1.txt file2.txt
```

    - To stage all changes, you can use a dot (`.`) to represent the current directory:

```
git add .
```

- To stage a single file, specify the file name:

```
git add file1.txt
```

4. **Check Repository Status:**
   - To see the status of your repository, use the `git status` command:It will show you the current branch, any modified or untracked files, and other relevant information.

```
git status
```

5. **Commit Changes:**
   - To save the staged changes, use the `git commit` command:Replace "Commit message" with a brief description of the changes you made. It's good practice to provide meaningful commit messages.

```
git commit -m "Commit message"
```

6. **Check Repository Log:**
   - To view the commit history and log, use the `git log` command:This will display a list of commits, including their unique identifiers, author information, dates, and commit messages.

```
git log
```

7. **Push Changes:**

- To push your committed changes to the remote repository, use the `git push` command:Replace `<branch-name>` with the name of the branch you want to push to (e.g., `main`). This will update the remote repository with your changes.

```
git push

 OR

 git push origin <branch-name>
```

8. **Move a File from Staging to Unstaged:**

- If you want to remove a file from the staging area and revert it to the previous state, use the `git restore` command:This will move the file from the staging area back to the working directory, keeping the previous version intact.

```
git restore --staged file1.txt
```

9. **Stash changes :**

To reapply a previously saved change from the stash and remove it from the stash, you can use the `git stash pop` command. This command will apply the most recent stash and remove it from the stash stack.

Here's the syntax:

```
git stash pop
```

This command will apply the changes from the topmost stash, merge them into your working directory, and remove the stash from the stash stack. Git will try to apply the stash on the branch you're currently on.

On the other hand, if you want to reapply a previously saved change from the stash but keep it in the stash as well, you can use the `git stash apply` command.

Here's the syntax:

```
git stash apply
```

This command applies the changes from the most recent stash to your working directory, but it keeps the stash intact. The stash remains available, allowing you to reapply it again in the future if needed.

Both `git stash pop` and `git stash apply` commands are useful when you want to bring back the changes you stashed earlier without losing them completely. The difference is that `git stash pop` removes the stash after applying, while `git stash apply` keeps the stash for later use.

10. **Git Branching**

    a. **What is branching ? and What are the different branches ?**

    The process of creating a separate line of development within a repository such that we diverge from the main development line, and make changes independently on our own branch, and then merge them back when ready.

    1. Master Branch:
        - The main branch of a repository.
        - Contains the stable and production-ready code.
        - Typically represents the latest release version of the software.

2. Hotfix Branch:
    - Created from the master branch.
    - Used to quickly address critical issues or bugs in the production code.
    - Isolated from other branches to allow for focused fixes.
    - Once the hotfix is completed, it is merged back into the master branch and may also be merged into other active branches if necessary.
3. Release Branch:
    - Created from the master branch.
    - Used to prepare the codebase for a new release.
    - Includes final testing, bug fixes, and any necessary last-minute changes.
    - Once the release is ready, it is merged into the master branch and given a version tag.
4. Feature Branch:
    - Created from the development branch (such as master or a release branch).
    - Used for developing new features or enhancements.
    - Allows developers to work on features independently without impacting the main codebase.
    - Once a feature is complete, it is merged back into the development branch, typically through a pull request or code review process.
5. Development Branch:
    - Also known as the "develop" branch.

- Serves as the integration branch for ongoing development work.
- Acts as a staging area for merging feature branches.
- Developers frequently push their feature branches to the development branch to ensure compatibility and resolve conflicts before merging into the main branch.

Note: The specific branch names and their purposes may vary depending on the development workflow or the project's conventions. The above descriptions provide a general understanding of commonly used branches in software development.

b. **Create a new Branch**

```
git branch <new-branch-name-here>
```

c. **Rename the branch :**

```
git branch -m <oldname> <newname>
```

d. **Go inside the branch or checkout the branch**

```
git checkout <branch-name>
```

e. **Create a branch as well as checkout at the same time**

```
git checkout -b <branch-name>
```

f. **After making changes to feature branch**

After making some changes on the feature branches if you have made some commits and stagd them already and now want to push the changes to the remote repo then use the below command

```
git push <brannch-name-where-to-push> <your-feature-branch-name>
```

After this pull request will be generated at the GitHub cloud platform which needs to be approved after approving the pull request the changes made inside the feature branch will be merged with the remote repository or the main branch now this changes will be only visible on the github cloud platform which is the remote repository of the main branch to be able to see the changes made on this report repository into a local repository we need to first pull the changes into our local main repository. So first checkout the main branch

```
git checkout main
```

and then pull the changes made on remote-repo to local-repo

```
git pull
```

g. to delete the branch that you have created :

first checkout to some different branch other than which you are deleting

```
git branch -d <branch-name-to-delete>
```

Now this will delete the branch from your local report but you also need to confirm it with GitHub server so you need to push this delete change on the remote repository

```
git push origin --delete <branch-name-to-delete>
```

h. **Making changes to the branch that has an open-pull request that has not been yet approved**

```
git push <remote> <your-branch> -f
```

i. **Do this Before pushing the changes to the main remote :**

If you want to incorporate the changes made in the main branch of the remote repository into your feature branch on your local repository, you can follow these steps:

1. **Switch to the main branch:** First, ensure you are on the main branch of your local repository. Use the command:

```
git checkout main
```

2. Pull the latest changes: Retrieve the latest changes from the main branch of the remote repository and update your local main branch. Use the command:

```
git pull origin main
```

3. Switch back to the feature branch: Move back to your feature branch where you want to include the changes from the main branch. Use the command:

```
git checkout feature1
```

4. Merge the main branch into the feature branch: Merge the changes from the main branch into your feature branch using the command:

```
git merge main
```

5. This will bring the changes from the main branch into your feature branch on your local repository.
6. Resolve any conflicts: If there are any conflicting changes between the main branch and your feature branch, you will need to resolve them manually. Git will indicate the conflicting files, and you can edit them to resolve the conflicts.
7. Commit the merged changes: Once you have resolved any conflicts, add and commit the merged changes in your feature branch using the usual Git commands:

```
git add .
 git commit -m "Merge changes from main branch"
```

8. Push the changes to the remote repository: Finally, push the merged changes from your feature branch to the remote repository:

```
git push origin feature1
```

By following these steps, you can include the changes made in the main branch of the remote repository into your feature branch on your local repository and then push the updated feature branch to the remote repository.

!!! THANK YOU !!!