**Lab#1 GIT**

**Basics:**

1. **What is version control? Why is it important?**

Version control is the practice of tracking and managing changes to files over time. It is a critical tool for software development which is used to manage any type of digital asset, such as documents, images, and videos. Version control systems track every change made to a file, including who made the change, when it was made, and what the change was.Version control is important for several reasons, including:

* It provides a safety net. Version control systems keep a complete history of all changes made to a file, so you can easily revert to a previous version if needed. This is essential for recovering from mistakes or undoing unwanted changes.
* It facilitates collaboration. Version control systems make it easy for multiple users to work on the same project simultaneously. Each user can have their own copy of the repository, and they can make changes without affecting other users' work. This can help to improve productivity and reduce the risk of conflicts.
* It promotes transparency. Version control systems make it easy to see who made what changes to a file, when they were made, and why. This can help to improve accountability and make it easier to track down the source of problems.
* It enables reproducibility. Version control systems make it easy to reproduce a specific version of a project. This is useful for debugging, testing, and deploying software.

1. **What is the difference between Git and GitHub?**

|  |  |
| --- | --- |
| **GIT** | **GITHUB** |
| Distributed version control system | Web-based hosting service for Git repositories |
| Open source | Privately owned by Microsoft |
| Free for public repositories, paid plans for private repositories | Free for individual users, paid plans for teams & organizations |
| Command-line interface, graphical user interface, web hooks, issue tracking | Graphical user interface, web hooks, issue tracking, code review, project management tools |

1. **Describe the Git workflow (add, commit, push, pull).**

The Git workflow is a set of steps that developers use to manage their code changes. The workflow typically involves the following steps:

Add: Add the files that you have changed to the staging area.

Commit: Create a snapshot of the staged changes.

Push: Push the commits to the remote repository.

Pull: Pull the latest changes from the remote repository.

1. **What is a repository in the context of Git?**

A repository in the context of Git is a collection of files and folders that are tracked by Git. A repository can contain any type of file, but it is most used for software development. Git repositories are organized into branches. A branch is a lightweight movable pointer to a commit. Branches allow you to work on different features or bug fixes without affecting the main codebase.

When you make changes to a file in a Git repository, Git creates a snapshot of the file. This snapshot is called a commit. Each commit represents a single change to your codebase. Git repositories can be stored locally on your computer or on a remote server. When you push changes to a remote repository, you are making those changes available to other developers.

**Commits:**

1. **What is a commit in Git?**

A commit in Git is a snapshot of your codebase at a specific point in time. It contains a list of all the changes that have been made since the previous commit. Commits are stored in a linear history, meaning that each commit points to the commit that came before it. This history allows you to track changes to your codebase over time.

Commits are made using the git commit command.

1. **How is each commit uniquely identified?**

Each commit in Git is uniquely identified by a SHA-1 hash. The SHA-1 hash is a 40-character hexadecimal string that is generated from the contents of the commit. The SHA-1 hash is a very secure hash function, which means that it is very unlikely that two different commits will have the same SHA-1 hash. This makes it possible to uniquely identify each commit in Git. For Example, we have a SHA-1 key: *123456781abcdef1234567890abcdef1234567890abcdef1234567890*

**Remote Repositories:**

1. **What is a remote repository in the context of Git?**

A remote repository in the context of Git is a Git repository that is hosted on a remote server. This allows you to share your code with other developers and collaborate on projects. Remote repositories can be hosted on a variety of different services, such as GitHub, GitLab, and Bitbucket. To create a remote repository on one of these services, you will need to create an account and create a new repository. They allow you to share your code with other developers and work on projects together.

1. **What are the default names that Git uses for the repository you cloned from and your local repository?**

The default name that Git uses for the repository you cloned from is origin. The default name that Git uses for your local repository is the name of the directory that you cloned the repository into. You can change the name of the remote repository using the git remote rename command. You can also change the name of your local repository by moving the directory that contains the repository.

1. **How do you synchronize changes from a remote repository to your local one, and vice versa?**

To synchronize changes from a remote repository to your local one, you can use the git pull command. This command will fetch the latest changes from the remote repository and merge them into your local repository. To synchronize changes from your local repository to the remote repository, you can use the git push command. This command will push your local changes to the remote repository.

**GitHub Specifics:**

1. **What is a pull request?**

A pull request is a way to propose changes to a codebase. It is a method used to request that changes made to a branch in a repository be merged into the main branch. PRs are typically used to request code review from other developers before merging changes. To create a PR, you must first create a branch in the repository where you want to make your changes. Once you have created a branch, you can make your changes and commit them to the branch. Once you are finished making your changes, you can create a PR to request that your changes be merged into the main branch.

1. **How do you 'fork' a repository on GitHub, and why might you want to?**

To fork a repository on GitHub, go to the repository page and click the Fork button in the top-right corner. This will create a copy of the repository in your own GitHub account. We want to do it - to contribute to the upstream repository, to create our own version of the repository and to learn from the repository.

1. **How can you use GitHub to collaborate on open-source projects?**

* Forking: Forking allows you to create a copy of a repository in your own GitHub account. This makes it easy to propose changes to the upstream repository without affecting the main codebase.
* Pull requests: Pull requests allow you to request that your changes be merged into the upstream repository. Other developers can review your changes and provide feedback before they are merged.
* Issues: Issues can be used to track bugs, feature requests, and other tasks. This makes it easy for developers to stay organized and work together to improve the project.
* Code review: GitHub provides a built-in code review tool that allows developers to comment on each other's code and provide feedback. This helps to ensure that the code is high quality and that everyone is on the same page.

**Collaboration and Best Practices:**

1. **Why is it important to write clear commit messages?**

* To help you and other developers understand what changes have been made to the codebase. When you look back at your commit history, you should be able to quickly understand what changes were made in each commit. This can be helpful for debugging, troubleshooting, and tracking down the source of bugs.
* To help other developers review your code. When you create a pull request, other developers will need to review your changes before they can be merged into the main codebase. Writing clear commit messages will help them to understand what changes you have made and why you have made them.
* To help future developers understand the codebase. As a codebase grows and changes over time, it can become difficult to understand what all the code does. Clear commit messages can help future developers to understand the history of the codebase and what each piece of code does.

1. **When collaborating with others, why might it be important to frequently pull the latest changes?**

* To avoid merge conflicts. When multiple developers are working on the same codebase, there is a risk of merge conflicts. This occurs when two developers make changes to the same file and then try to merge their changes. By frequently pulling the latest changes, you can reduce the risk of merge conflicts.
* To stay up-to-date with the latest changes. When you are working on a collaborative project, it is important to stay up-to-date with the latest changes that other developers have made. By frequently pulling the latest changes, you can ensure that you are working on the most recent version of the code.
* To avoid accidentally overwriting other developers' work. If you are not frequently pulling the latest changes, you may accidentally overwrite other developers' work. By frequently pulling the latest changes, you can ensure that you are not overwriting other developers' work.
* To resolve conflicts early. If merge conflicts do occur, it is best to resolve them as early as possible. By frequently pulling the latest changes, you can identify and resolve merge conflicts early on.
* To work efficiently. When you are working on a collaborative project, it is important to work efficiently. By frequently pulling the latest changes, you can avoid wasting time working on changes that have already been made by other developers.