# **BASICS OF GIT AND GITHUB**

Git is a version control system (VCS) that track changes in files and data over time. Imagine it like a time machine for your code or project files. You can rewind to any previous version, see what changed, and even revert to an earlier state if needed. This is super helpful for developers, as it allows them to:

* Collaborate on projects with multiple people without worrying about conflicts or overwriting each other's work.
* Experiment with new features or ideas without fear of breaking everything.
* Keep track of progress and see how a project has evolved over time.
* Easily revert to previous versions if something goes wrong.

GitHub is a popular online platform that offers Git repositories for hosting and managing the code. Think of it as a big storage locker for all your Git repositories. It lets you:

* Store your code publicly or privately (depending on your preference).
* Share your code with others and collaborate on projects.
* Track changes and contributions made by different people.
* Access your code from anywhere with an internet connection.

Simple analogy to understand the difference between Git and GitHub:

Think of Git as a notebook where you track changes to your project files. Think of GitHub as a cloud storage service where you can save and share your notebook with others.

## **GITHUB TERMS**

Here's an introduction to some basic terms used in GitHub:

**Repository (Repo):**It's the heart of GitHub, a virtual storage space for your project's files and code. ️Think of it as a central folder where all the project's components are organized and tracked.

# **Commit:**A snapshot of your project's changes at a specific point in time. It's like saving a checkpoint in a game, allowing you to revert back to that version if needed.

# **Branch:**A separate line of development, allowing you to experiment with new features or fixes without affecting the main project. It's like creating a parallel universe for your code, where you can safely test new ideas.

# **Issue:**A task, bug report, or discussion thread related to the project. It's like a sticky note on the virtual whiteboard, helping to track progress and keep everyone on the same page.

# **Fork:**A personal copy of someone else's repository, allowing you to experiment with their code without affecting the original project. ️ It's like making a blueprint of a building so you can modify it without changing the actual structure.

# **Pull Request (PR):**A request to merge changes from one branch into another, often used for code reviews and collaboration. It's like asking for permission to incorporate your work into the main project.

**Clone:** Creating a local copy of a repository on your computer, enabling you to work on it offline. It's like downloading a project onto your machine so you can tinker with it locally.

**Push:** sending your local changes to the remote repository on GitHub, making them available to others.It's like uploading your work to the cloud so others can see it and contribute.

**Pull:** Fetching the latest changes from the remote repository to your local machine, keeping your local copy up-to-date.It's like downloading the latest version of a project to ensure you have the most recent advancements.

**BASIC COMMANDS OF GIT:**

* **Git config command**: This command configures the user. The Git config command is the first and necessary command used on the Git command line. This command sets the author name and email address to be used with your commits. Git config is also used in other scenarios.
* **Git init command:**This command is used to create a local repository.
* **Git clone command:** This command is used to make a copy of a repository from an existing URL.
* **Git add command:** This command is used to add one or more files to staging (Index) area.

## **CONCEPTS OF GITHUB , GITLAB , BITBUCKET**

All three platforms serve the same general purpose: hosting and managing Git repositories for software development. They offer similar features like version control, code collaboration, issue tracking, and project management.

* **GitHub:** Ideal for open-source projects, large communities, and beginners due to its extensive documentation and resources.
* **GitLab:** Perfect for teams that utilize the entire DevOps lifecycle and require project management tools beyond just code hosting.
* **Bitbucket:** Best for enterprises that need secure private repositories, Atlassian integration, and high collaboration features.

**INDUSTRIAL PRACTICES OF USING GIT:**

1. Feature Branching Strategy:Teams move away from the "master branch" mentality and embrace feature branching. Each developer creates a dedicated branch for their feature or bug fix, isolating their work from the main codebase. This avoids conflicts and facilitates independent development.

2. Frequent Commits and Descriptive Messages:Commits are not reserved for milestones; small, atomic changes are encouraged, with clear and concise commit messages explaining the rationale behind the changes. This improves traceability and simplifies code reviews.

3. Code Reviews and Pull Requests:Before merging changes, code reviews are mandatory. Pull requests allow other developers to review the code, suggest improvements, and ask questions. This fosters collaboration and ensures high code quality.

4. Continuous Integration and Deployment (CI/CD):Git seamlessly integrates with CI/CD pipelines. Automated builds, tests, and deployments catch errors early and accelerate the release cycle. This improves code stability and allows for faster iterations.

5. Standardized Workflow and Branch Naming Conventions:Teams establish clear conventions for branch naming, commit messages, and workflow stages. This ensures consistency, eases collaboration, and promotes a streamlined development process.

6. Access Control and Permissions:Role-based access controls grant different levels of permission to team members. This ensures sensitive code is protected while facilitating collaboration for authorized individuals.

7. Branch Protection and Merge Guidelines:Specific rules can be set for branches, like requiring code reviews or prohibiting direct pushing to certain branches. This prevents accidental merges and maintains code quality.

8. Documentation and Training:Clear documentation and regular training on Git best practices are vital for onboarding new team members and ensuring everyone shares the same understanding of the workflow.

9. Version Control Tools and Extensions:Tools like GitKraken or GitHub Desktop provide user-friendly interfaces and visualizations to enhance the Git experience. Extensions for code reviews, linting, and other tasks further streamline the workflow.

10. Automated Tasks and Scripts:Repetitive tasks like setting up branches, running tests, or deploying code can be automated using scripts. This frees up developers' time for more valuable work.

**Steps to clone a Git repository to local machine:**

1. Obtain the Repository URL

2. Open a Terminal or Command Prompt

3. Execute the git clone Command

4. Authenticate (if necessary)

RESOURCES USED: ww3schools,Javapoint,github docs