Git

Git is a distributed version control system that allows you to track changes in your codebase over time.

Here are the fundamental concepts and commands to get started:

Initialize a new Git repository: git init
Stage changes for commit: git add <file>
Commit changes: git commit -m "Commit message"
View commit history: git log
Check repository status: git status

Branching and Merging:
Branching and merging are essential aspects of Git that enable parallel development and the integration of changes. Here are the key commands and practices:

Create a new branch: git branch <branch_name>
Switch to a branch: git checkout <branch_name>
Merge branches: git merge <branch_name>
Resolve merge conflicts: Manually edit conflicted files, then git add <file> and git commit

Branching Strategy:

Git branching strategy refers to the approach and guidelines for creating and managing branches in a Git repository. It helps teams collaborate effectively, organize development efforts, and maintain a stable codebase. There are several popular branching strategies, but I will describe two commonly used ones: Gitflow and GitHub Flow.

Gitflow:
Gitflow is a branching model that provides a structured workflow for larger projects with multiple releases and long-term maintenance. It consists of two main branches, master and develop, along with supporting branches for feature development and bug fixes.

master branch: The master branch represents the stable codebase. It only contains production-ready code and should ideally be deployable at any time. Tagging is often used to mark specific points in the project history, representing releases.

develop branch: The develop branch acts as an integration branch for ongoing development. It serves as a parent branch for all feature and bug branches. When new features or bug fixes are completed, they are merged back into the develop branch.

Feature branches: When working on a new feature, developers create a feature branch from the develop branch. They work on the feature in isolation until it is complete and then merge it back into the develop branch.

Release branches: When preparing for a release, a release branch is created from the develop branch. This branch undergoes final testing, bug fixing, and any necessary release-specific adjustments. Once the release is ready, it is merged into both master and develop branches, and the release branch is closed.

Hotfix branches: If a critical bug is found in the production code, a hotfix branch is created from the master branch. The bug is fixed in the hotfix branch, which is then merged into both master and develop branches. The changes are also backported to the develop branch to ensure the fix is included in future releases.

GitHub Flow:
GitHub Flow is a simpler branching model that is well-suited for smaller teams and projects with frequent deployments. It revolves around a single main branch, typically master or main, and emphasizes small, incremental changes.

master branch: The master branch in GitHub Flow represents the production-ready code. It should always be in a deployable state, reflecting the latest stable version of the project.

Feature branches: When working on a new feature or bug fix, developers create a feature branch from the master branch. They make their changes and push the branch to the remote repository. After the feature is complete, a pull request is created, and the changes are reviewed by the team. Once approved, the branch is merged into master and deployed.

Continuous Integration (CI): GitHub Flow encourages the use of continuous integration tools to automatically build, test, and deploy code changes as soon as they are merged into the master branch. This ensures that the main branch is always in a deployable state.

Releases: Releases in GitHub Flow are typically represented by tags. When a set of features and bug fixes are ready for deployment, a tag is created on the master branch to mark the release point.

It's important to note that these are just two examples of branching strategies, and there are other approaches and variations depending on the specific needs of a project or team. The choice of branching strategy should be based on factors such as team size, project complexity, release frequency, and deployment requirements.

The above branching strategy is very important. Kindly read the below shared blog also.

https://www.abtasty.com/blog/git-branching-strategies/

Remote Repositories:
Remote repositories allow collaboration and sharing of code with others. Here's what you need to know:

Clone a remote repository: git clone <repository_url>
Push changes to a remote repository: git push origin <branch_name>
Pull changes from a remote repository: git pull origin <branch_name>

Collaboration Workflows:

Collaboration workflows in Git refer to the practices and strategies teams use to work together efficiently. Here are two common workflows:

Forking Workflow:

Create a fork of a repository, make changes in your fork, and submit a pull request to propose your changes to the original repository. This is useful for contributing to open-source projects or collaborating on projects with distributed teams.

Branching Workflow (e.g., GitFlow or GitHub Flow):

Develop each feature or task on a separate branch. Start by creating a branch from the main branch. After completing the work, merge the feature branch back into the main branch. This allows for parallel development, easy isolation of features, and straightforward integration.

Git Tools:
Git provides command-line tools and graphical user interface (GUI) tools to work with repositories:

Command-line Tools (e.g., Git Bash): A text-based interface that allows you to interact with Git through commands.

Graphical Tools (e.g., GitKraken, SourceTree): GUI tools provide a visual interface for managing repositories, visualizing the commit history, and resolving conflicts.

Please go through below shared links

https://www.youtube.com/watch?v=SWYqp7iY_Tc

Bit bucket: https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud

Git Cheat sheet: https://cs.fyi/guide/git-cheatsheet