Git Basics Handbook

I. Introduction to Version Control

A. Definition and Significance of Version Control Systems

- Version control systems (VCS) track changes to files over time, allowing multiple contributors to work on a project simultaneously while preserving the history of changes.

B. Benefits of Utilizing Version Control for Software Development

- Collaboration: Facilitates teamwork by providing a centralized platform for managing code.
- Versioning: Maintains a history of changes, enabling easy rollback to previous states.
- Traceability: Tracks who made which changes, aiding in debugging and accountability.

II. Core Concepts of Git

A. Repositories: Local and Remote

- Local Repository: A copy of the project on your local machine, containing the entire version history.
- Remote Repository: A shared repository stored on a server, allowing collaboration among multiple developers.

B. Working Directory: Workspace for Project Files

- The directory on your local machine where you modify files for your project.

C. Staging Area (Index): Selecting Changes for Commits

- An intermediate area where changes are prepared before being committed to the repository.

D. Commits: Capturing Project States with Descriptive Messages

- A snapshot of the project at a specific point in time, accompanied by a descriptive commit message.

E. Branches: Divergent Development Paths within a Repository

- Independent lines of development that allow for parallel work on different features or bug fixes.

III. Essential Git Commands

A. Initialization: Creating a New Git Repository

- git init: Initializes a new Git repository in the current directory.

B. Tracking Changes: Identifying Modified Files

- git status: Displays the status of modified files in the working directory.

C. Staging and Committing: Preparing and Recording Changes

- git add <file>: Adds file changes to the staging area.
- git commit -m "message": Records staged changes with a descriptive message.

D. Branching: Creating and Switching Between Development Lines

- git branch <branch_name>: Creates a new branch.
- git checkout <branch_name>: Switches to the specified branch.

E. Merging: Integrating Changes from Different Branches

- git merge <branch_name>: Combines changes from the specified branch into the current branch.

IV. Mastering Git Workflows

A. Feature Branch Workflow: Streamlined Development and Integration

- Create a new branch for each feature or bug fix.
- Regularly merge feature branches into the main development branch (e.g., master or main).

B. Gitflow Workflow: Structured Approach for Large-Scale Projects

- Utilizes different types of branches (e.g., feature, develop, release, hotfix) for managing development stages.
- Offers a more structured approach suitable for complex projects with multiple contributors.

V. Advanced Git Techniques

A. Resolving Merge Conflicts: Handling Conflicting Changes

- Identify conflicting changes during a merge operation and resolve them manually.

B. Stashing Changes: Temporarily Shelving Uncommitted Work

- git stash: Temporarily stores changes that are not ready to be committed, allowing you to switch branches or apply fixes.

C. Using Tags: Annotating Specific Project Versions

- git tag <tag_name>: Creates a tag to mark a specific commit (e.g., for releases or milestones).