**Git and GitHub: A Comprehensive Guide**

**Introduction**

Git is a distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers collaboratively developing source code during software development. GitHub is a web-based hosting service for Git repositories. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features.

**I. Git Basics**

* **Version Control:**
  + Git allows you to track changes to your code, revert to previous versions, and collaborate effectively with others.
  + It maintains a history of all changes, enabling you to understand how a project evolved.
* **Repositories:**
  + A Git repository is a storage location for your project's files and their history.
  + Local repositories reside on your computer, while remote repositories are hosted on servers like GitHub.
* **Workflow:**
  + The typical Git workflow involves:
    - Making changes to files.
    - Staging those changes.
    - Committing the staged changes.
    - Pushing commits to a remote repository.

**Commands:**

1. **git init**:
   * Initializes a new Git repository in the current directory.
   * Creates a .git subdirectory containing repository metadata.
   * git init my\_project: Initializes a new git repository in a folder named my\_project.
2. **git clone <repository\_url>**:
   * Copies a remote repository to your local machine.
   * Creates a working copy of the entire repository.
   * git clone https://github.com/user/repo.git
3. **git config --global user.name "Your Name"**:
   * Sets your name for Git commits globally.
4. **git config --global user.email "your.email@example.com"**:
   * Sets your email address for Git commits globally.
5. **git config --list**:
   * Displays your Git configuration settings.
6. **git status**:
   * Shows the status of changes in your working directory and staging area.
   * Indicates modified, staged, and untracked files.

**II. Working with Files**

* **Staging Area (Index):**
  + The staging area is an intermediate area where you prepare changes for a commit.
  + It allows you to selectively include changes in your next commit.
* **Commits:**
  + A commit is a snapshot of your repository at a specific point in time.
  + Each commit has a unique identifier (SHA-1 hash) and a commit message.

**Commands:**

1. **git add <file>**:
   * Adds a specific file to the staging area.
   * git add index.html
2. **git add .**:
   * Adds all changes in the current directory to the staging area.
3. **git commit -m "Commit message"**:
   * Records staged changes to the repository with a descriptive message.
4. **git commit --amend -m "Updated commit message"**:
   * Edits the last commit.
5. **git rm <file>**:
   * Removes a file from the working directory and staging area.
6. **git rm --cached <file>**:
   * Removes a file from the staging area, but not the working directory.
7. **git mv <old\_file> <new\_file>**:
   * Renames a file.
8. **.gitignore**:
   * Creates a file named .gitignore, add file names or file patterns inside the file. Git will ignore those files. This is very important to avoid commiting unnecessary files like node\_modules, or .env files.

**III. Viewing History**

* **Logs:**
  + Git logs provide a detailed history of commits, including commit messages, authors, and timestamps.
* **Differences:**
  + Git diff allows you to see the changes between different versions of your files.

**Commands:**

1. **git log**:
   * Displays the commit history.
2. **git log --oneline**:
   * Displays a concise, one-line commit history.
3. **git log --graph**:
   * Displays the commit history as a graph.
4. **git diff**:
   * Shows changes between the working directory and staging area.
5. **git diff --staged**:
   * Shows changes between the staging area and the last commit.
6. **git show <commit>**:
   * Displays the content and metadata of a specific commit.

**IV. Branching and Merging**

* **Branches:**
  + Branches allow you to work on different features or bug fixes in parallel.
  + They create isolated lines of development.
* **Merging:**
  + Merging combines changes from one branch into another.
* **Rebasing:**
  + Rebasing is an alternative to merging that creates a linear commit history.

**Commands:**

1. **git branch**:
   * Lists all local branches.
2. **git branch <branch\_name>**:
   * Creates a new branch.
3. **git checkout <branch\_name>**:
   * Switches to the specified branch.
4. **git checkout -b <branch\_name>**:
   * Creates and switches to a new branch.
5. **git merge <branch\_name>**:
   * Merges the specified branch into the current branch.
6. **git branch -d <branch\_name>**:
   * Deletes a local branch (if it's already merged).
7. **git branch -D <branch\_name>**:
   * Forces the deletion of a branch.
8. **git rebase <branch\_name>**:
   * Reapplies commits on top of another base branch.

**V. Remote Repositories and GitHub**

* **Remote Repositories:**
  + Remote repositories are hosted on servers like GitHub, GitLab, or Bitbucket.
  + They enable collaboration and backup of your code.
* **GitHub Features:**
  + Pull requests: Used for code review and collaboration.
  + Issues: Used for tracking bugs, feature requests, and tasks.
  + Forking: Creates a copy of a repository in your GitHub account.
  + GitHub Actions: Used for CI/CD.

**Commands:**

1. **git remote add origin <repository\_url>**:
   * Adds a remote repository named "origin".
2. **git remote -v**:
   * Lists remote repositories and their URLs.
3. **git push origin <branch\_name>**:
   * Pushes local commits to the remote repository.
4. **git push -u origin <branch\_name>**:
   * Sets upstream tracking information.
5. **git pull origin <branch\_name>**:
   * Fetches and merges changes from the remote repository.
6. **git fetch origin**:
   * Fetches changes from the remote repository without merging.

**VI. Stashing and Resetting**

* **Stashing:**
  + Stashing allows you to temporarily save changes that are not ready for commit.
* **Resetting:**
  + Resetting allows you to revert to previous commits.

**Commands:**

1. **git stash**:
   * Temporarily saves changes that are not ready for commit.
2. **git stash list**:
   * Lists stashed changes.
3. **git stash apply**:
   * Applies the most recent stashed changes.
4. **git stash pop**:
   * Applies and removes the most recent stashed change.
5. **git reset HEAD <file>**:
   * Unstages a file.
6. **git reset --hard <commit>**:
   * Resets the repository to a previous commit, discarding all changes.
7. **git revert <commit>**:
   * creates a new commit that reverts the changes that were made in the specified commit.