Git Workflow Summary

Christophe Troalen

October 23, 2024

1 Introduction

Git is a distributed version control system that allows multiple developers to collaborate on a project by tracking changes to files over time. This document summarizes the essential concepts of Git, the workflow involving branches, and the use of various Git commands. Understanding these concepts is crucial for effective collaboration in software development.

2 Key Concepts

2.1 Repository and Branches

- **Repository:** A Git repository is a storage space for your project, which contains all files, directories, and the entire history of changes. It can exist locally on your machine or remotely on a platform like GitHub or GitLab.

  A Git repository consists of two main parts: the local repository (where your code and its history are stored on your machine) and the remote repository (which is often hosted on platforms like GitHub, GitLab, or Bitbucket).

  When you make a commit, you are recording changes to the files in your local repository. This action creates a new commit object that captures the current state of your project, including the changes made and the associated commit message.

- **Branching:** The default branch in a repository is usually called `main` (or `master`). Branches are essential for isolating development work. Developers create feature branches to work independently on new features or bug fixes without affecting the stable code in the main branch.

2.2 Basic Commands

Here are some key Git commands you'll frequently use:

- `git init` - Initializes a new Git repository in the current directory. It creates a hidden `.git` directory to store all version control information.

- `git clone <repository-url>` - Clones an existing repository from a remote location to your local machine. This command creates a copy of the repository and sets up a remote called `origin`.
- `git branch <branch-name>` - Creates a new branch. This allows you to work on a new feature without affecting the main branch. You can also list branches with `git branch`.
- `git checkout <branch-name>` - Switches to the specified branch. This command updates your working directory to reflect the state of that branch.
- `git add .` - Stages all changes (new, modified, deleted files) for the next commit. You can also specify individual files to add.
- `git commit -m "message"` - Commits the staged changes to the repository with a descriptive message. This message helps track the history of changes.
- `git push origin <branch-name>` - Pushes your local commits to the remote repository associated with the specified branch. This command updates the remote branch with your local changes.
- `git pull` - Fetches changes from the remote repository and merges them into your current branch. This is a combination of `git fetch` and `git merge`.
- `git fetch` - This part retrieves updates from the remote repository but does not modify your local files or branches. It updates your remote-tracking branches, such as origin/main.
- `git diff main origin/main` - After fetching, you can compare your local branches with the updated remote tracking branches. For example, you can see what changes have been made in the origin/main branch compared to your local main branch by using.
- `git merge` - This part attempts to merge the changes fetched from the remote into your current branch. If there are changes on the remote that your current branch does not have, Git will try to merge them.

3 The `git fetch` Command

The `git fetch` command is used to update your local repository with changes from the remote repository without merging them into your current branch. Here are its main features:

- **Updates Remote Tracking Branches:** When you run `git fetch`, Git downloads new commits, branches, and tags from the remote repository. It updates your remote tracking branches (e.g., `origin/main`) to reflect the latest state of the remote repository.

- **Does Not Affect Your Working Directory:** `git fetch` does not modify your current branch or working directory. Your local files remain unchanged, allowing you to review changes before deciding to merge.
- **Review Before Merging:** After fetching, you can compare your local branches with the updated remote tracking branches. This allows you to understand the changes made in the remote repository and address any potential conflicts before they affect your work.

3.1 Example Workflow with `git fetch`

Here's a typical workflow that includes `git fetch`:

1. **Fetch changes:**

   ```
   git fetch origin
   ```

   This command retrieves updates from the remote repository and updates your remote tracking branches.

2. **Check for updates:** After fetching, you can check which branches have been updated by running:

   ```
   git branch -r
   git log origin/main --oneline
   ```

   This allows you to see what changes have been made in the remote repository.

3. **Merge changes into the local branch:** If you decide to incorporate the changes into your local branch, switch to your local `main` branch and merge the changes:

   ```
   git checkout main
   git merge origin/main
   ```

   Alternatively, if you want to keep your local commit history linear, you might choose to rebase instead:

   ```
   git rebase origin/main
   ```

4 Conclusion

The `git fetch` command is an essential tool for developers working in collaborative environments. It allows teams to stay updated with the latest changes from the remote repository while maintaining control over their local branches. By following a systematic workflow, developers can effectively manage their contributions to the codebase and minimize conflicts.