Basic overview of Git and GitHub

Basics of Git and GitHub, step by step

Step 1: Understanding Git

Git is a distributed version control system that allows developers to track changes in their code over time. It helps manage collaborative coding projects efficiently and allows developers to work on code individually and merge changes seamlessly.

Step 2: Installation and Setup

First, make sure you have Git installed on your computer. You can download and install Git from the official website: https://git-scm.com/downloads

After installation, open your terminal (for Mac and Linux) or Git Bash (for Windows) and configure your name and email using the following commands:

```
git config --global user.name "Your Name"
git config --global user.email "youremail@example.com"
```

Step 3: Creating a Git Repository

To start using Git, you need to create a Git repository. A repository (or repo) is a folder that tracks changes to your project files. You can either initialize a new repository or clone an existing one from a remote location (like GitHub).

To initialize a new repository, navigate to your project folder in the terminal and run:

```
git init
```

Step 4: Adding and Committing Changes

After making changes to your files, you need to stage them for committing. Staging is the process of selecting which changes you want to include in the next commit. To stage changes, use:

```
git add <file>   # To stage a specific file
git add .        # To stage all changes in the current directory
```

Once the changes are staged, you can commit them with a descriptive message:

```
git commit -m "Your commit message here"
```

Step 5: Branching

Branching allows you to work on different features or bug fixes without affecting the main codebase. The default branch is usually called "master" or "main." To create a new branch, use:

```
git branch <branch-name>   # Creates a new branch
git checkout <branch-name> # Switches to the new branch
```

Step 6: Merging

After you've completed your changes on a branch, you can merge it back into the main branch to incorporate your work. Switch to the target branch (e.g., master) and use:

```
git merge <branch-name>
```

Step 7: GitHub Collaboration

GitHub is a web-based platform that hosts Git repositories and provides additional collaboration features. You can create a new repository on GitHub and push your local repository to it using the following commands:

```
git remote add origin <repository-url>   # Associates your local repo with the
remote one
git push -u origin <branch-name>         # Pushes your branch to the remote
repository
```

You can also pull changes from the remote repository using:

```
git pull origin <branch-name>
```

Step 8: Pull Requests

Pull Requests (PRs) are a way to propose changes from a branch in a repository on GitHub. Team members can review the changes and discuss before merging them into the main branch.

To create a pull request, go to your repository on GitHub, select your branch, and click on the "New pull request" button.

Working with a team on Git and GitHub

Working with a team on Git and GitHub can greatly enhance collaboration and make managing codebase changes more efficient.

Here are some best practices and tips for working with your team on Git and GitHub:

1. **Branching Strategy**: Agree on a branching strategy for your team. One popular approach is the Gitflow workflow, where you have a "master" branch for production-ready code, a "develop" branch for ongoing development, and feature branches for individual features or bug fixes. This helps isolate changes and makes it easier to manage releases.
2. **Pull Requests (PRs)**: Encourage the use of pull requests when team members want to introduce changes to the main codebase. PRs allow for code review, feedback, and discussion before merging changes. This helps maintain code quality and ensures that everyone is aware of the modifications being made.
3. **Code Review**: Perform thorough code reviews on pull requests. Code reviews not only catch potential bugs but also promote knowledge sharing and adherence to coding standards within the team. Encourage constructive feedback and use tools like GitHub's inline comments for specific code suggestions.
4. **Commit Messages**: Encourage descriptive and meaningful commit messages. Clear commit messages make it easier for team members to understand changes, track progress, and troubleshoot issues. Use imperative language in commit messages (e.g., "Add feature" instead of "Added feature").
5. **Git Workflow Education**: Ensure that all team members are familiar with the Git commands and workflow. Provide training or documentation if needed. This ensures that everyone is on the same page and follows the agreed-upon practices.
6. **Regular Pulls/Pushes**: Regularly pull changes from the remote repository to keep your local repository up to date. Likewise, push your changes frequently to avoid conflicts and ensure that everyone has access to the latest codebase.
7. **Handling Conflicts**: Conflicts can occur when multiple team members make changes to the same file or lines of code. If a conflict arises, communicate with the team members involved to resolve it. Review changes together and decide on the best resolution.
8. **Code Style and Linting**: Agree on coding style guidelines and use linters or code formatters to enforce consistency. This ensures that the codebase remains readable and maintainable by all team members.
9. **Use Git Hooks**: Git hooks allow you to automate tasks before or after certain Git events. For example, you can use pre-commit hooks to run tests or code linting before making a commit.
10. **Issue Tracking**: Integrate your Git repository with an issue tracking system (e.g., GitHub Issues, Jira). This helps in tracking and managing tasks, bugs, and features effectively.
11. **Security and Access**: Ensure that access to the repository is appropriately managed. Limit access to sensitive branches or settings to authorized team members only. Regularly review access rights to maintain security.
12. **Documentation**: Encourage documentation of significant changes, new features, or any complex parts of the codebase. This helps new team members understand the project and serves as a reference for the future.

A step-by-step demonstration of some of the practices using Git commands and GitHub examples.

Let's go through a sample workflow:

Step 1: Create a Repository on GitHub

- Go to https://github.com/ and log in to your GitHub account.

- Click on the "New" button to create a new repository.
- Provide a repository name, description, and choose the repository's visibility (public or private).
- Click on the "Create repository" button.

Step 2: Clone the Repository to Your Local Machine

In your terminal or Git Bash, navigate to the directory where you want to store the local copy of the repository and run the following command:

```
git clone <repository-url>
```

Replace `<repository-url>` with the URL of the repository you just created on GitHub.

Step 3: Set Up the Branching Strategy

In this example, we'll use the Gitflow workflow as the branching strategy.

- Initialize Gitflow on your local repository:

```
git flow init
```

Follow the prompts to set up the default branch names.

Step 4: Create and Work on a Feature Branch

- Create a new feature branch:

```
git flow feature start add-new-feature
```

- Make changes to your code in the feature branch and commit the changes:

```
# Make your changes to the code
git add .
git commit -m "Add new feature: [Description of your changes]"
```

Step 5: Push the Feature Branch to GitHub

```
git push origin feature/add-new-feature
```

Step 6: Create a Pull Request (PR)

- Go to your GitHub repository.

- You should see a message saying "Your recently pushed branches" with a button to "Compare & pull request" for the "add-new-feature" branch. Click on it.
- Provide a title and description for the pull request, and then click on the "Create pull request" button.

Step 7: Review the Pull Request

In a real team environment, other team members would review the code changes in the pull request, leave comments, and provide feedback. For this demo, let's assume you are reviewing your own pull request.

Step 8: Merge the Pull Request

- If everything looks good, click on the "Merge pull request" button.
- Confirm the merge and add a descriptive commit message.
- Click on the "Confirm merge" button.

Step 9: Update Your Local Repository

Now, let's update your local repository with the latest changes from the remote repository:

```
git checkout develop
git pull origin develop
```

Step 10: Finish the Feature Branch

```
git flow feature finish add-new-feature
```

Step 11: Push the Develop Branch to GitHub

```
git push origin develop
```

Demostration of the Palmfit Gitflow

The flow you provided outlines the steps for working with Git and Gitflow in a team environment.

Let's break down each step

1. **Clone the repository**: To start working with the project, team members clone the remote repository to their local machines using the command:

```
git clone <repository-url>
```

This creates a local copy of the entire repository, including all branches and commit history.

2. **Checkout to your new branch**: Before starting work on a new feature or bug fix, it's essential to create a new branch and switch to it using the command:

```
git checkout -b users/firstName.LastName/feature/api-for-Abc
```

This command creates a new branch named `users/firstName.LastName/feature/api-for-Abc` and switches to it. Branch names often follow a naming convention that includes the developer's name, feature name, or issue identifier.

3. **Checkout to develop branch**: To keep your local repository up-to-date, switch back to the `develop` branch using:

```
git checkout develop
```

4. **Pull from develop**: Pull the latest changes from the `develop` branch to your local `develop` branch using:

```
git pull origin develop
```

This ensures that you have the most recent changes made by other team members in the `develop` branch.

5. **Checkout to your branch**: Return to your working branch before merging the latest changes from `develop` into it using:

```
git checkout users/firstName.LastName/feature/api-for-Abc
```

6. **Merge develop to your current branch**: Merge the changes from the `develop` branch into your current branch (feature branch) using the merge command:

```
git merge develop
```

This step incorporates the latest changes from `develop` into your feature branch, ensuring it's up-to-date with the latest developments in the project.

7. **Push to your branch and create a pull request (PR)**: Once you've resolved any conflicts and are satisfied with your changes, push your branch to the remote repository:

```
git push origin users/firstName.LastName/feature/api-for-Abc
```

This command pushes your changes to the remote repository, making them available for others to see.

After pushing, you can create a pull request on the GitHub platform. Pull requests allow other team members to review your changes, provide feedback, and ultimately merge your branch into the `develop` or `master` branch.

This Gitflow workflow ensures that team members work on their respective branches, regularly integrate changes from the `develop` branch to their feature branches, and collaborate effectively using pull requests. It helps manage changes and ensures that the main development branches remain stable and in sync with the latest code changes.