Version Control with Git and GitHub

Version control is an essential skill for any developer or DevOps engineer. It allows teams to track changes to code, collaborate efficiently, and ensure that past versions of code can be retrieved if needed. In this session, we will explore **Git**, the most popular version control system, and **GitHub**, a platform that provides cloud-based Git repository hosting and collaboration tools.

Overview of Git and GitHub

Git:

- **Git** is a **distributed version control system** designed to track changes in your source code.
- It allows multiple developers to work on the same codebase without interfering with each other's work.
- Changes are saved in **snapshots** (also known as commits), so you can always revert to a previous version.

GitHub:

- **GitHub** is a cloud-based platform where Git repositories can be hosted.
- It provides collaboration tools, such as **pull requests**, **issue tracking**, and **continuous integration**.

Getting Started with Git

Step 1: Installing Git

You can install Git on any operating system:

- **Linux**: Use your package manager (e.g., `sudo apt install git` for Ubuntu).
- **Windows/Mac**: Download Git from git-scm.com.

To verify the installation, run:

```
git --version
```

Step 2: Configuring Git

After installation, you need to set up your user information. Git attaches this information to each commit:

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

You can check your configuration at any time using:

```
git config --list
```

Key Git Concepts

1. **Repository (Repo)**: This is the project folder that Git will manage. It can be local or hosted on platforms like GitHub.

2. **Working Directory**: Your local file system where you work on files.
3. **Staging Area**: A place where changes are reviewed before committing.
4. **Commit**: A snapshot of your staged changes.
5. **Branch**: A parallel version of your code, useful for working on features independently.
6. **Merge**: The act of combining changes from different branches.
7. **Pull/Push**: Pull is getting the latest changes from a remote repository, and Push is sending your local changes to the remote repository.

Basic Git Workflow

To understand Git, let's follow a simple workflow. We will create a repository, track changes, commit them, and push them to GitHub.

1. **Initializing a Local Repository**

   The first step is to create a Git repository in your project folder:

   ```
   mkdir my-first-repo
   cd my-first-repo
   git init
   ```

   This creates an empty Git repository in the folder, and Git starts tracking your project.

2. **Tracking Changes in Files**

   Next, let's create a simple file and track it using Git.

   ```
   echo "Hello, Git!" > hello.txt
   ```

At this point, Git is aware that a new file (`hello.txt`) has been added, but it's not yet being tracked.

You can check the status of your repository using:

```
git status
```

Output:

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
     hello.txt
```

3. **Staging Changes**

   To start tracking a file, you need to stage it using the `git add` command:

   ```
   git add hello.txt
   ```

   Now, when you run `git status`, Git will show the file as ready to be committed:

   ```
   Changes to be committed:
     (use "git restore --staged <file>..." to unstage)
       new file: hello.txt
   ```

4. **Committing Changes**

   After staging, you can commit the changes. A commit saves the current state of your project.

```
git commit -m "Added hello.txt"
```

The `-m` flag is used to provide a commit message. This message should describe what was changed in the commit.

5. **Viewing Commit History**

   You can view the history of commits using:

```
git log
```

Output:

```
commit a1b2c3d4e5 (HEAD -> master)
Author: Your Name <youremail@example.com>
Date: Mon Oct 4 14:00:00 2024

    Added hello.txt
```

Working with GitHub

Step 1: Creating a GitHub Repository

- Go to [GitHub]() and sign up (if you haven't already).
- Once signed in, click the **New** button to create a new repository.
- Give your repository a name (e.g., `my-first-repo`) and click **Create Repository**.

Step 2: Connecting Your Local Repo to GitHub

Now that you have a local repository and a remote repository on GitHub, you need to link them. First, copy the **remote URL** from your GitHub repository page. Then, in your local repository, run:

```
git remote add origin
https://github.com/yourusername/my-first-repo.git
```

This adds a **remote** repository named `origin` that points to your GitHub repo.

6. Pushing Changes to GitHub

Now, push your local commits to GitHub:

```
git push -u origin master
```

The `-u` flag sets `origin` as the default remote, so you can just use `git push` next time.

7. Cloning a GitHub Repository

If you want to clone a repository from GitHub to your local machine, use:

```
git clone https://github.com/yourusername/my-first-repo.git
```

This downloads a copy of the repository and sets it up locally.

Working with Branches

- **Creating a Branch:** Branches allow you to work on new features or bug fixes without affecting the main codebase.

```
git checkout -b new-feature
```

This command creates and switches to a new branch called `new-feature`.

- **Making Changes in the Branch**

  Now, you can make changes in your branch:

```
echo "New feature!" > feature.txt
git add feature.txt
git commit -m "Added feature.txt"
```

- **Merging Branches**

  Once you've finished your work, you can merge the `new-feature` branch into the `master` branch:

  a. First, switch back to the `master` branch:

```
git checkout master
```

  b. Then merge your feature branch:

```
git merge new-feature
```

- **Deleting a Branch**

  After merging, you can delete the `new-feature` branch:

```
git branch -d new-feature
```

Collaborating with Others on GitHub

GitHub provides several collaboration tools that allow teams to work together efficiently:

1. **Forking a Repository:** Forking a repository creates a personal copy of another user's project under your GitHub account. It allows you to experiment with changes without affecting the original project.

   To fork a project:

   - Navigate to the project repository on GitHub.
   - Click the **Fork** button in the top-right corner.

2. **Creating a Pull Request**: A **pull request** is how you propose changes to a repository. Once changes are made in your branch, you can open a pull request to merge these changes back into the original repository.

   Here's how you can create a pull request:

   - Push your changes to a new branch on GitHub.
   - Go to the repository on GitHub and click **Compare & pull request**.
   - Provide a description of the changes and submit the pull request.

Pro Tips for Using Git and GitHub

- **Commit Often**: Each commit should represent a logical unit of work.
- **Write Descriptive Commit Messages**: Commit messages should describe the changes succinctly.
- **Use Branches for Features**: Always create a new branch for each new feature or bug fix.
- **Collaborate via Pull Requests**: Pull requests make it easier for teams to review and discuss code before merging.
- **Resolve Merge Conflicts Carefully**: When merging branches, you might encounter conflicts. Use `git status` and GitHub's conflict resolution tools to resolve them.

Conclusion

Version control with Git and GitHub is a critical skill for developers. It enables efficient collaboration, code management, and project tracking. By following these steps and practicing, you will gain confidence in managing your code and working in teams. The next step is to deepen your knowledge by working on real-world projects with others on GitHub!