Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant

GitHub

A brief guide

Mohammad Idrees Bhat

Introduction to GitHub

GitHub is a platform that simplifies collaboration and version control, primarily for software development projects, though it can be used for any kind of collaborative project.

It builds upon Git, the most widely used distributed version control system, and provides a web-based interface that enhances Git's capabilities with features such as bug tracking, project management, continuous integration (CI), and much more.

Key Concepts:

1. Version Control:

- A way to track changes in your files and revert to specific versions when necessary. It allows multiple people to collaborate on a project without overriding each other's work.

2. Git:

- Git is a distributed version control system that lets you manage and track the history of your code. It's used to handle the collaboration between team members and enables efficient project management.
- GitHub is a hosting platform for repositories created with Git. It adds extra functionality like a web interface and tools for team collaboration.

3. Repository (Repo):

- A repository is like a project folder that contains all of your project's files and the complete revision history of each file. A repository can either be stored locally on your computer or remotely on GitHub.

4. Commit:

- A commit is like a snapshot of your repository at a specific point in time. Every time you make changes and want to save them, you create a commit with a message explaining what was changed.

5. Branch:

- A branch is an independent line of development. You can use branches to work on different features, bug fixes, or experiments without affecting the main part of your project. Once the work is completed, you can merge it back into the main branch.

6. Fork:

- A fork is a personal copy of another user's repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

7. GitHub Projects:

- GitHub Projects is a feature that allows you to organize and track the progress of your work in a project board, similar to a Kanban board. It integrates with GitHub Issues and Pull Requests to streamline project management.

Commonly Used Commands

- `git config --global user.name "Your Name"`
  Sets the global username for Git.
- `git config --global user.email "you@example.com"`
  Sets the global email for Git.
- `git init`
  Initializes a new Git repository.
- `git clone <repo-url>`
  Clones a repository from a remote URL.
  Example: `git clone https://github.com/username/repo.git`
- `git branch`
  Lists, creates, or deletes branches.
- `git checkout <branch>`
  Switches to a different branch.
  Example: `git checkout feature-branch`
- `git merge <branch>`
  Merges the specified branch into the current branch.
- `git rebase <branch>`
  Reapplies commits on top of another branch.
- `git add <file>`
  Stages a file for the next commit.

- `git commit -m "message"`
  Commits staged changes with a message.
- `git status`
  Shows the current state of the working directory and staging area.
- `git diff`
  Displays changes between the working directory and the last commit.
- `git log`
  Shows a log of all commits made in the repository.
- `git push`
  Pushes commits to a remote repository.
- `git pull`
  Fetches and merges changes from the remote repository.
- `git fetch`
  Downloads changes from a remote repository without merging.
- `git remote -v`
  Lists all remote connections (URLs of repositories).
- `git stash`
  Temporarily saves uncommitted changes.
- `git stash pop`
  Restores previously stashed changes.
- `git reset <file>`
  Removes a file from the staging area.
- `git rm <file>`
  Deletes a file from both the working directory and the Git index.

Setting Up Git and GitHub

Before you start using GitHub, you need to install and configure Git on your machine. Follow these essential steps to get up and running quickly.

1. Installing Git

- **Windows**: Download Git from Git for Windows and follow the installation instructions.
- **macOS**: Use Homebrew in the terminal:
  `brew install git`
- **Linux**: Install git using:
  ```
  sudo apt-get install git  # Ubuntu/Debian
  sudo yum install git      # CentOS/Fedor
  ```

2. Configuring Git

- Set your user information:

  ```
  git config --global user.name "Your Name"
  git config --global user.email "your_email@example.com"
  ```
- Verify the configuration with:

```
git config --list
```

This command will display all your current Git configuration settings, including your username, email, and any other settings you've configured globally or locally. It's a useful way to confirm that Git is set up correctly on your machine.

Creating repository

You can create a GitHub repository directly from the GitHub website or from the command line using Git. Here's how:

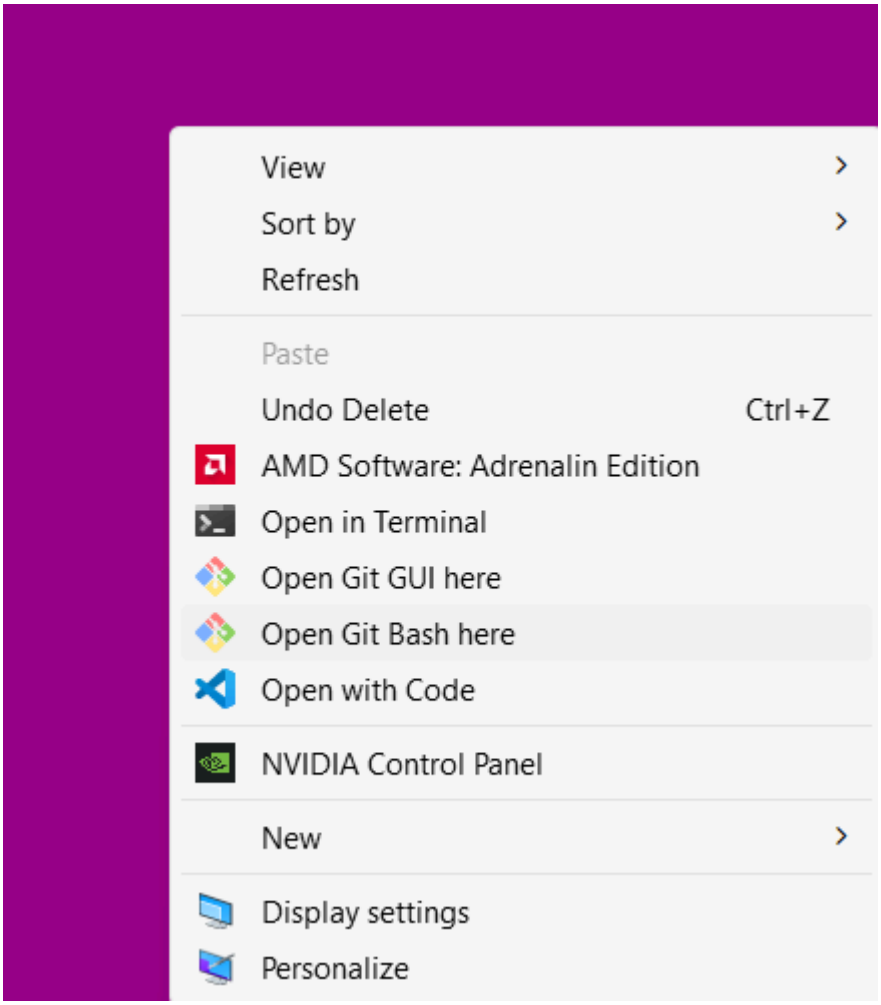
Method 1: Creating a Repository on GitHub (Website)

- **Step 1: Go to GitHub**
  Log in to your GitHub account and navigate to the **Repositories** tab.
- **Step 2: Click "New"**
  Click the **New** button to create a new repository.
- **Step 3: Fill in Repository Details**
  - **Repository name**: Choose a descriptive name for your project.
  - **Description (optional)**: Add a short description of your project.
  - **Public/Private**: Select whether the repository will be public (anyone can see it) or private (only you and people you invite can see it).
  - **Initialize with a README**: You can choose to automatically create a README file, which is usually the main entry point for your project documentation.
- **Step 4: Create the Repository**
  Click **Create repository** to finish.

Method 2: Creating a Repository Using Git (Locally)

If you prefer to create a repository on your local machine and push it to GitHub later, follow these steps. You'll need to have Git installed and a terminal (or Git Bash for Windows) to execute these commands.

1. **Create a new directory locally**
   Navigate to the folder where you want to create your project and create a new directory. Use git bash for this activity, which must be installed on your system. This step will create a folder on your system.




```
mkdir new-repo-name // use a name for this folder
   cd new-repo-name // enter your newly made folder
```

2. **Initialize a Git repository**

   Initialize Git in this new directory:

   ```
   git init
   ```
   This command creates a hidden .git folder, marking it as a Git repository.

3. **Add your project files**

If you already have files in the folder, you can add them to the repository:

```
git add .
```

4. **Commit the files**

If you already have files in the folder, you can add them to the repository:

```
git commit -m "Initial commit"
```

5. **Link to a remote GitHub repository**

Now, you need to link your local repository to a remote GitHub repository. First, create a repository on GitHub (without initializing it with any files). Once the repository is

created, you'll get the repository URL. Add it as the remote origin for your local repo:

`git remote add origin git@github.com:yourusername/repo-name.git`
You can replace the yourusername/repo-name.git with the repository link that you get from GitHub.

6. **Push your changes to GitHub**

Push your local repository to the remote GitHub repository:

`git push -u origin master`

Pushing and Pulling Changes to GitHub

Pushing and pulling changes are crucial Git operations that allow you to sync your local repository with a remote repository (like GitHub).

1. Pushing Changes to GitHub

1. **Commit Your Local Changes**

   - Before pushing, make sure all your changes are committed locally:
     `git add .`
     `git commit -m "Commit message"`
   - `git add .` stages all modified files for commit.
   - `git commit -m "Commit message"` commits those changes with a message describing what was changed.
2. **Push to GitHub**

   - To push your changes to the remote repository on GitHub, run:
     `git push origin master`
     - `origin`: Refers to the default name of the remote repository.
     - `master`: The branch you're pushing to (replace `master` with the appropriate branch name if necessary).

---

2. Pulling Changes from GitHub

1. **Pull Changes from GitHub**

   - To fetch and merge changes from the remote repository into your local repository, use:
     `git pull origin master`
     - This command pulls updates from the remote `master` branch and merges them into your local repository.
2. **Resolve Conflicts (If Any)**

   - If there are conflicts between the remote and local changes, Git will notify you, and you'll need to manually resolve them before committing the changes:
     `git add .`
     `git commit -m "Resolved conflicts"`

- **Push**: Sends your local changes to the remote repository on GitHub.
- **Pull**: Fetches changes from the remote repository and merges them into your local repository.

Most commonly used Git commands

1. Configuration Commands

These commands set up your Git environment and are critical before starting any Git project.

- `git config --global user.name "Your Name"` - Sets the global username.
  Example: `git config --global user.name "John Doe"`
- `git config --global user.email "you@example.com"` - Sets the global email.
  Example: `git config --global user.email "john@example.com"`

---

2. Repository Setup and Initialization

These commands help initialize and clone repositories, setting up your workspace.

- `git init` - Initializes a new Git repository.
- `git clone <repo-url>` - Clones a repository from a remote URL.
  Example: `git clone https://github.com/username/repo.git`

---

3. Branching and Merging

These commands allow you to manage different versions of your project using branches and merge them as needed.

- `git branch` - Lists, creates, or deletes branches.
  Example: `git branch` (List), `git branch new-branch` (Create)
- `git checkout <branch>` - Switches to a specific branch.
  Example: `git checkout feature-branch`
- `git merge <branch>` - Merges a branch into the current branch.
  Example: `git merge feature-branch`
- `git rebase <branch>` - Reapplies commits on top of another branch.
  Example: `git rebase main`

---

4. Making Changes (Staging and Committing)

These commands help you track changes to your files and prepare them for saving in the repository.

- `git add <file>` - Stages a file for the next commit.
  Example: `git add filename.txt`
- `git commit -m "message"` - Commits staged changes with a message.
  Example: `git commit -m "Initial commit"`

---

5. **Viewing and Managing Changes**

These commands allow you to check the status of changes and view commit history.

- `git status` - Shows the current state of the working directory and staging area.
- `git diff` - Shows changes between commits, branches, or working directory.
- `git log` - Displays the commit history.

---

6. **Remote Repositories (Syncing with Others)**

These commands help manage connections to remote repositories and fetch/push changes.

- `git push` - Pushes commits to a remote repository.
  Example: `git push origin main`
- `git pull` - Fetches and merges changes from the remote repository.
  Example: `git pull origin main`
- `git fetch` - Downloads changes from a remote repository without merging.
  Example: `git fetch origin`
- `git remote -v` - Lists all remote repository connections.

---

7. **Stashing and Resetting**

These commands allow you to temporarily save work or remove files from the staging area.

- `git stash` - Temporarily saves uncommitted changes.
- `git stash pop` - Restores stashed changes.
- `git reset <file>` - Removes a file from the staging area.
  Example: `git reset filename.txt`

---

8. **File Management**

These commands deal with file operations in Git.

- `git rm <file>` - Deletes a file from both working directory and Git index.
  Example: `git rm filename.txt`

- Another good resource - A visual Git Guide

https://marklodato.github.io/visual-git-guide/index-en.html

THANK YOU!

Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant