A Beginner's Guide to Git and GitHub for Hacktoberfest

Your Journey to Open-Source Contribution Starts Here

Enigma - Git & GitHub Workshop

September 2025

Abstract

This document is a step-by-step guide for absolute beginners to get started with Git and GitHub. It covers installation, configuration, local repository management, and the complete workflow for making a contribution to an open-source project during Hacktoberfest.

Contents

1 Introduction: What are Git, GitHub, and Hacktoberfest?

Before we dive into commands, let's understand the tools.

- **Git**: A free and open-source *version control system*. It's a tool that runs on your computer and helps you track changes in your code. Think of it as a series of "snapshots" (called commits) of your project that you can revisit anytime.
- **GitHub**: A web-based platform for hosting Git repositories. It adds a social and collaborative layer on top of Git, allowing developers to work together, review code, and manage projects. It's where you'll find projects to contribute to.
- **Hacktoberfest**: An annual, month-long event in October that encourages people to contribute to open-source projects. By making valid contributions (pull requests), you can earn cool swag and become part of the global developer community.

2 First-Time Setup: Getting Your Tools Ready

This section covers the one-time setup on your local machine. These commands are for Debian-based Linux distributions (like Ubuntu). Run these on the terminal.

2.1 Install Git and GitHub CLI

The GitHub Command Line Interface (`gh`) is a tool that brings GitHub to your terminal, making the contribution process smoother.

```
# Install Git
sudo apt install git

# Install GitHub CLI
sudo apt install gh
```

2.2 Verify Installation

Check that both tools were installed correctly by checking their versions.

```
git --version
gh --version
```

2.3 Initial Git Configuration

These global settings will apply to every Git repository on your system.

```
# Sets the default branch name to 'main' for any new repository you create.
# This is the modern standard, replacing the old 'master'.
git config --global init.defaultBranch main

# Creates a handy alias 'tree' to view your commit history in a compact,
    graphical way.
git config --global alias.tree "log --oneline --decorate --graph --all"
```

To use the alias we just created, you'll simply type `git tree` in your terminal inside a Git repository. It will be shown later.

3 The Local Workflow: Your Personal Sandbox

Let's practice Git in a safe, local environment.

3.1 Create a Local Repository

```
# Create a directory to hold your projects
mkdir local_repo
cd local_repo

# Create a new project folder
mkdir myProject
cd myProject

# Initialize an empty Git repository in this folder
# This creates a hidden .git directory to track changes.
git init
```

3.2 Configure Your Identity

Git needs to know who is making the commits. It's best to set this locally for each project, but you can set it globally if you use the same identity everywhere.

- **Important:** You must first have a GitHub account. Let's assume your username is `myUsername` and your email is `myEmail`.

```
# Set your name and email for THIS repository only (--local flag)
git config --local user.name "myUsername"
git config --local user.email "myEmail"

# --- ALTERNATIVELY: For global configuration ---
# If you want to use the same identity for all your projects:
# git config --global user.name "myUsername"
# git config --global user.email "myEmail"
```

3.3 Authenticate with GitHub

Connect your terminal to your GitHub account using the GitHub CLI. This will make interacting with remote repositories much easier.

```
# This command will open a browser window to authorize the CLI.
gh auth login
```

3.4 The Core Git Cycle: Add, Commit, Branch

This is the fundamental workflow you'll use constantly.

3.4.1 Add & Modify Files

Create some files in your `myProject` directory. For example, a `README.md` and a `hello.txt`. After creating or changing files, you tell Git which changes you want to save. This is called **staging**.

Your most used command should be `git status`. It tells you the current state of your repository, showing any new (untracked) files or modified files.

```
# Check the status of your repository. Use this command often!
# It will show ‘untracked files’ in red.
git status

# Stage a specific file to be included in the next commit.
# After this, ‘git status’ will show this file as a ‘change to be committed’ in
    green.
```

```
git add "README.md"

# To stage all new and modified files in the current directory:
# git add .

# To unstage a file (move it from staged back to modified):
# git restore --staged "README.md"
```

3.4.2 Commit Changes

A **commit** is a snapshot of your staged changes at a specific point in time. A good commit message is crucial for understanding the project's history.

```
# Commit the staged files with a short message and a longer description
git commit -m "feat: Add initial project files" -m "Created README.md and hello
    .txt as starting point."

# DANGER ZONE: To undo the last commit and all its changes:
# Use with extreme caution as it discards work permanently.
# git reset --hard HEAD^
```

3.4.3 Viewing Your History

Now that you've made a commit, you can view the project's history. The standard command is `git log`. However, our `git tree` alias provides a much cleaner, graphical view.

```
# View the detailed commit history
git log

# View the history with our compact, graphical alias
git tree
```

3.4.4 Branching

A **branch** is an independent line of development. You should *always* create a new branch for a new feature or bug fix. This keeps your main branch clean.

```
# Create a new branch called 'newBranch'
git branch newBranch

# Switch your working directory to the 'newBranch'
git checkout newBranch

# --- SHORTCUT: Create and checkout a new branch in one command ---
# git checkout -b newBranch

# Make some more changes, then add and commit them on this new branch.
# For example, edit hello.txt, then:
# git add hello.txt
# git commit -m "docs: Update hello.txt with new info"

# To delete a branch (use with care):
# git branch -D newBranch
```

You can now use `git tree` to see your commit history. Notice how the graph shows your new branch pointing to the latest commit!

```
# The output will visually show 'newBranch' pointing to your new commit
# and 'main' pointing to the commit before it.
git tree
```

4 Hacktoberfest Contribution Workflow

Now, let's apply these skills to contribute to a real project.

4.1 Register for Hacktoberfest

Since it is currently October, now is the perfect time to register!

1. Go to the official Hacktoberfest website: https://hacktoberfest.com/
2. Click "Register" and authorize with your GitHub account.
3. You're all set! Now your contributions during October will be tracked.

4.2 The Remote Contribution Cycle

This workflow involves finding a project on GitHub, making a copy, changing it, and proposing your changes.

4.2.1 Step 1: Fork the Repository

A **fork** is your personal cloud copy of someone else's project. You can't push changes directly to projects you don't own, so you fork them first.

- Find a project on GitHub with the 'hacktoberfest' topic. Let's say it's located at `OWNER/REPO`.

```
# Go back to your main projects directory
cd ~/local_repo

# Use gh to fork the repository to your GitHub account
gh repo fork OWNER/REPO
```

4.2.2 Step 2: Clone Your Fork Locally

Cloning means downloading a copy of a repository from the cloud (your fork) to your local machine.

```
# Clone the repository you just forked (replace myUsername)
git clone https://github.com/myUsername/REPO.git

# --- SHORTCUT: Fork and clone in one go ---
# gh repo fork OWNER/REPO --clone

# Navigate into the newly cloned project directory
cd REPO
```

4.2.3 Step 3: Create a New Branch for Your Feature

Never work directly on the `main` branch.

```
git checkout -b newFeature
```

4.2.4 Step 4: Make, Add, and Commit Your Changes

This is where you do the actual work! Fix a bug, add a feature, improve documentation, etc.

```
# ... make your code changes here ...

# Check the status to see a list of your modified files
git status

# Stage all your changes
git add .

# Commit your changes with a clear message
git commit -m "feat: Add new awesome feature" -m "This feature does X and Y,
    fixing issue #123."
```

4.2.5 Step 5: Push Your Branch to Your Fork

Send your new branch and its commits from your local machine up to your forked repository on GitHub. `origin` is the default name for the remote URL you cloned from.

```
git push origin newFeature
```

4.2.6 Step 6: Create a Pull Request (PR)

A **Pull Request** is a proposal to the original project maintainers to merge the changes from your branch into their main project. This is the final step of your contribution.

```
# gh makes this super easy. It will open an interactive prompt.
gh pr create
```

Follow the prompts: fill in a title, a detailed description of your changes, and submit. Congratulations, you've made a Hacktoberfest contribution!

5 Keeping Your Fork in Sync

The original project (`OWNER/REPO`) will continue to be updated by others. Your fork will become outdated. It's good practice to sync it before starting new work. Here is a simple way to do it.

1. **Sync Your Fork on the GitHub Website:**

   First, go to your forked repository on the GitHub website (e.g., `https://github.com/myUsername/REPO`). If your fork is behind the original project, GitHub will show you a message like ”This branch is X commits behind OWNER:main.”

   - Click the **'Sync fork'** button that appears. This will automatically update your fork on GitHub with the latest changes from the original project.

2. **Update Your Local Repository from Your Fork:**

   Now that your fork on GitHub is up-to-date, you need to bring those changes down to your local machine. You can do this with a single command.

   ```
   # First, make sure you are inside your local REPO folder
   cd REPO

   # Switch to your main branch
   git checkout main

   # Pull the latest changes from your GitHub fork (origin)
   git pull origin main

   ```

Now, your local repository is also in sync! From here, you can follow the same contribution workflow as before:

1. Create a new branch: `git checkout -b newFeature`
2. Make your changes and commit them.
3. Push your new branch to your fork: `git push origin newFeature`
4. Create a pull request.