Git and GitHub Workflow - A Step-by-Step Guide

1. Git Basics

1.1 Initialize a Git Repository

The first step is to set up a Git repository in your local project directory.

- **Command**: `git init`
- **Explanation**: This command initializes a new Git repository in your project folder. Git will now track changes made to the project.

Example:

```
shub@ubuntu:~/git_for_devops$ git init
Initialized empty Git repository in /home/shub/git_for_devops/.git/
shub@ubuntu:~/git_for_devops$
```

1.2 Stage Changes with `git add`

After making changes to files in your project, stage the files for commit.

- **Command**: `git add <file>` or `git add .` (to stage all changes)
- **Explanation**: This command adds the specified files to the staging area, preparing them for the next commit.

Example:

```
shub@ubuntu:~/git_for_devops$ ls
info.txt
shub@ubuntu:~/git_for_devops$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
shub@ubuntu:~/git_for_devops$ git add info.txt
```

Shubham Ahire

1.3 Commit Changes with `git commit`

Once your changes are staged, the next step is to save them as a commit.

- **Command**: `git commit -m "Your message"`
- **Explanation**: This command records the changes with a message describing what was changed or added.

Example:

```
shub@ubuntu:~/git_for_devops$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   info.txt

shub@ubuntu:~/git_for_devops$ git commit -m "new commit"
[master (root-commit) fffd2b0] new commit
 1 file changed, 1 insertion(+)
 create mode 100644 info.txt
shub@ubuntu:~/git_for_devops$
```

1.4 View Commit History with `git log`

To see the history of commits in the repository:

- **Command**: `git log`
- **Explanation**: This shows all past commits, their hashes, author details, and messages.
- If you don't want to see all the detailed views, use **git log - - oneline** for a more compact view.

Example:

```
shub@ubuntu:~/git_for_devops$ git log
commit fffd2b0efcee0ea81c194a12f9fc681ac769c276 (HEAD -> master)
Author: shubham ahire <shubham.cyberwhite@gamil.com>
Date:   Tue Oct 22 04:52:31 2024 +0000

    new commit
shub@ubuntu:~/git_for_devops$
```

1.5 Check Status with `git status`

At any point, you can check the current state of your working directory:

- **Command**: `git status`
- **Explanation**: This command tells you which files are staged, unstaged, or untracked, and whether there are pending changes for commit.

Example:

```
shub@ubuntu:~/git_for_devops$ git status
On branch master
nothing to commit, working tree clean
shub@ubuntu:~/git_for_devops$
```

1.6 HEAD Pointer

- **Explanation**: `HEAD` represents the current snapshot of your branch. It's important for tracking where you are in your project's history.
- HEAD is the **latest commit** made to a branch, there are multiple HEAD for every branch we have.

You can check which commit `HEAD` is pointing to:

```
shub@ubuntu:~/git_for_devops$ git log
commit fffd2b0efcee0ea81c194a12f9fc681ac769c276 (HEAD -> master)
Author: shubham ahire <shubham.cyberwhite@gamil.com>
Date:   Tue Oct 22 04:52:31 2024 +0000

    new commit
shub@ubuntu:~/git_for_devops$
```

---

2. Git Branching

2.1 Create a New Branch with `git branch`

- **Command**: `git branch <branch-name> (create empty branch)`
- **Explanation**: Branching is crucial for working on features independently without affecting the main branch. This command creates a new branch.
- It will create a new empty branch, whereas git checkout -b will create and switch to a new branch with all existing contents from the master from which it is created.

Example:

```
shub@ubuntu:~/git_for_devops$ git branch
* master
shub@ubuntu:~/git_for_devops$ git checkout -b dev
Switched to a new branch 'dev'
shub@ubuntu:~/git_for_devops$ git status
On branch dev
nothing to commit, working tree clean
```

2.2 Switch Branches with `git checkout`

- **Command**: `git checkout <branch-name>`
- **Explanation**: This command is used to switch to a different branch in your repository.
- As in the previous example, I'm on the **dev branch** after git checkout -b, now use **git checkout master** to switch to **master branch**.

Example:

```
shub@ubuntu:~/git_for_devops$ git branch
* dev
  master
  new_dev
shub@ubuntu:~/git_for_devops$ git checkout master
Switched to branch 'master'
```

2.3 Create and Switch in One Step with `git checkout -b`

- **Command**: `git checkout -b <new-branch-name>`
- **Explanation**: This command creates a new branch and switches to it in a single command.

Example:

```
shub@ubuntu:~/git_for_devops$ git checkout -b dev
Switched to a new branch 'dev'
shub@ubuntu:~/git_for_devops$ git status
On branch dev
nothing to commit, working tree clean
```

2.4 Merge Branches with `git merge`

- **Command**: `git merge <branch-name>`
- **Explanation**: After completing your work in a branch, you can merge it back into another branch (usually `main` or `master`).

Example:

```
shub@ubuntu:~/git_for_devops$ git checkout master
Switched to branch 'master'
shub@ubuntu:~/git_for_devops$ git merge dev
Updating fffd2b0..8b6bdf9
Fast-forward
 info.txt | 1 +
 1 file changed, 1 insertion(+)
```

2.5 Switch Branches Using `git switch`

- **Command**: `git switch <branch-name>`
- **Explanation**: An alternative and more intuitive command for switching branches.
- It is the same as the **git checkout branch**, both can be used to switch between branches.

Example:

bash
Copy code

```
git switch main
```

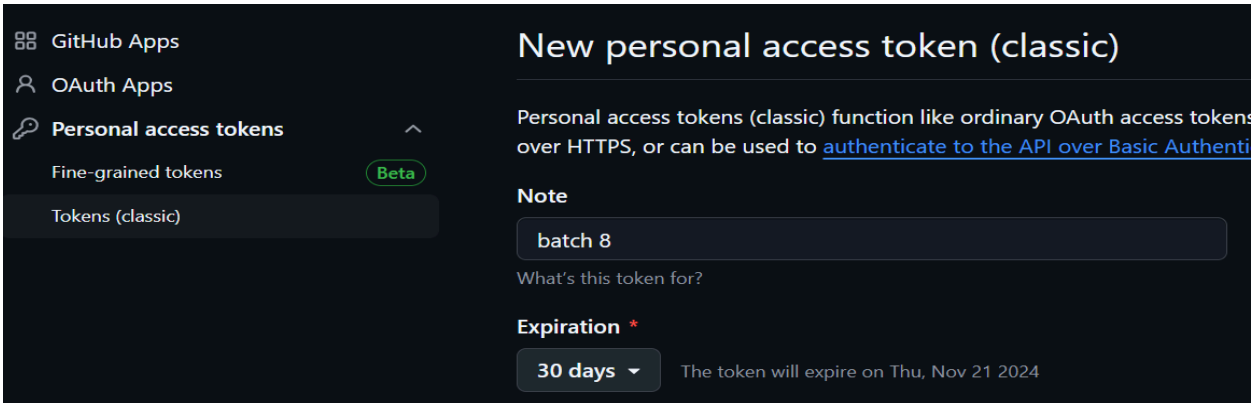
3. Connecting and Syncing with GitHub

3.1 Connect to GitHub using a Personal Access Token (PAT)

To push code to GitHub, you need to authenticate using a PAT.

- **Step 1**: Create a new PAT from GitHub by navigating to **Settings > Developer Settings > Personal Access Tokens > create token classic**.
- **Step 2**: Clone the remote repository first & use PAT when pushing code for the first time.
- **Step 3**: Checkgit remote -v, now paste the http code like **https://<token>@github.com/Shubham-Ahire/practice.git** now git push

Example:




```
ubuntu@ip-172-31-36-119:~/practice$ git remote -v
origin  https://github.com/Shubham-Ahire/practice.git (fetch)
origin  https://github.com/Shubham-Ahire/practice.git (push)
ubuntu@ip-172-31-36-119:~/practice$ git remote set-url origin https://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@github.com/Shubham-Ahire/practice.git
ubuntu@ip-172-31-36-119:~/practice$
```

```
ubuntu@ip-172-31-36-119:~/practice$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 320 bytes | 320.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Shubham-Ahire/practice.git
   14ab127..1df0237  master -> master
ubuntu@ip-172-31-36-119:~/practice$
```

3.2 Connect via SSH

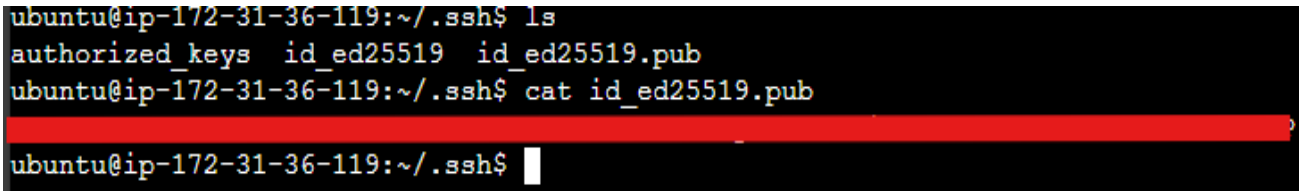
Setting up SSH is a secure way to interact with GitHub.

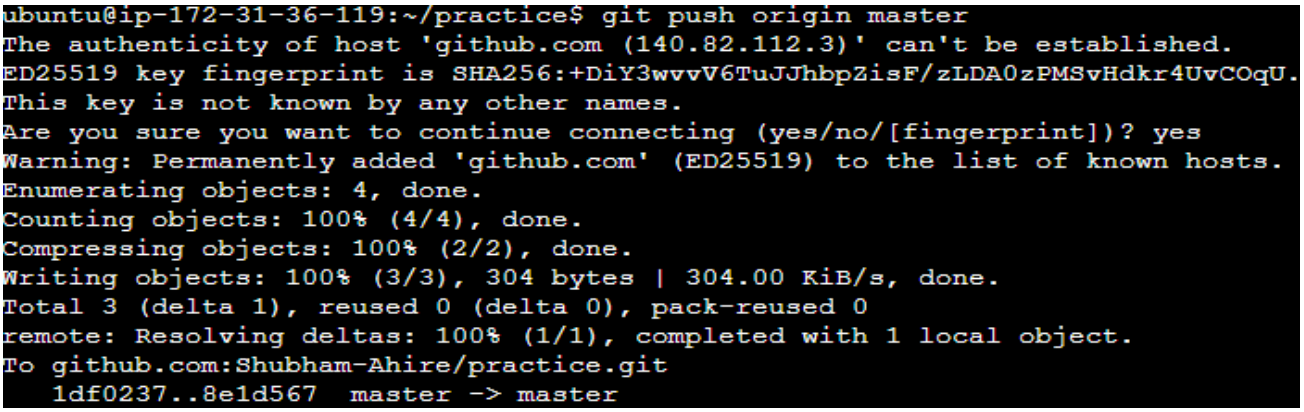
- **Step 1**: Generate an SSH key.

`ssh-keygen:` This will create a private & public key for you.

- **Step 2:** Now copy the public key (.pub) and then paste it as described below.
- **Step 3**: Add the SSH key to your GitHub account by copying the public key (`id_rsa.pub`) and adding it to **GitHub > Settings > SSH and GPG keys**.
- **Step 4:** Now set the remote -v, by set-url as told in case of PAT tokens.

Example:






```
git remote add origin git@github.com:username/repository.git
git push origin master
```

3.3 Push Code to GitHub

- **Command**: `git push`
- **Explanation**: Push your local commits to the GitHub repository.

Example:

```
ubuntu@ip-172-31-36-119:~/practice$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 320 bytes | 320.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Shubham-Ahire/practice.git
   14ab127..1df0237  master -> master
ubuntu@ip-172-31-36-119:~/practice$
```

3.4 Pull Code from GitHub

- **Command**: `git pull`
- **Explanation**: Retrieve the latest changes from the remote GitHub repository and merge them into your local branch.

Example:

```
ubuntu@ip-172-31-36-119:~/practice$ git pull origin master
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 932 bytes | 932.00 KiB/s, done.
From github.com:Shubham-Ahire/practice
 * branch            master     -> FETCH_HEAD
   8e1d567..4bb299f  master     -> origin/master
Updating 8e1d567..4bb299f
Fast-forward
 test.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
```

`git pull origin master`

4. Handling Pull Requests

4.1 Create a Pull Request (PR)

Once changes are pushed to GitHub, you can create a PR to propose these changes to be merged into a base branch (e.g., `main`).

Steps:

1. Push your branch to GitHub.
2. Go to GitHub, select your branch, and click **New Pull Request**.
3. Add a description and submit for review.

4.2 Code Review and Approval

To maintain code quality, it's crucial to review PRs before merging.

- **Tool**: **Coderabbit**
    - Review the changes proposed in the PR.
    - Coderabbit can assist with suggesting improvements and ensuring best practices.

4.3 Merge a Pull Request

Once the PR is approved, you can merge the branch into the `main` branch.

- Click on **Merge Pull Request** in GitHub.

After merging, clean up by deleting the feature branch.