Git &GitHub for DevOps

TEJAS POKALE

What is Git?

- A tool to track and manage code changes.
- Helps multiple people work on the same code without overwriting each other's work.

What is GitHub?

- A website where developers can store and share their code.
- It uses Git to track changes, allowing multiple people to collaborate on the same project.
- Includes features like issue tracking, pull requests, and code review for better collaboration.
- GitHub is a platform for hosting, managing, and collaborating on software projects.

Why Git is Important in DevOps:

- Version Control: Keeps track of all code changes.
- Branching: Allows developers to work on features separately.
- Merging: Combines everyone's work into one version.

Basic Git Commands:

- Start a Repo: git init
- Copy a Repo: git clone <URL>
- Save Changes:
    - Stage: git add <file>
    - Commit: git commit -m "message"
- Share Code: git push
- Get Updates: git pull

1. Installing Git

Windows

- Download Git from git-scm.com.

Linux

- Use the package manager for your distribution:
- sudo apt-get install git # For Debian/Ubuntu
- sudo yum install git # For CentOS/Red Hat

2. Configuring Git

Set up user information for commits:

git config --global user.name "Tejas"

git config --global user.email "your_email@example.com"

To verify the configuration:

git config –list

3. Personal Access Token (PAT) in GitHub

1. **Log in**: Go to GitHub and sign in.
2. **Open Settings**: Navigate to Profile > **Settings** > **Developer settings** > **Personal access tokens** > **Tokens (classic)**.
3. **Generate Token**: Click **Generate new token**, set a name, expiration, and required scopes (e.g., repo).
4. **Copy Token**: Save the token immediately (it won't be shown again).
5. **Use Token**: When prompted for a password in Git, use the token instead.
6. **Store Token** *(Optional)*: Save it securely using:
7. git config --global credential.helper manager

4. Initializing a Repository

Creating a New Repository

git init

Cloning an Existing Repository

git clone <repository_url>

5. Checking Repository Status

View the current status of your working directory and staging area:

git status

6. Adding and Committing Changes

Staging Changes

- Add a specific file:
- git add <file>
- Add all files:
- git add .

Committing Changes

Save changes with a descriptive message:

git commit -m "Your commit message"

7. Branching and Merging

Creating a New Branch

git branch <branch_name>

Switching to a Branch

git checkout <branch_name>

Merging a Branch

git merge <branch_name>

8. Working with Remote Repositories

Adding a Remote Repository

git remote add origin <repository_url>

Pushing Changes

git push origin <branch_name>

Pulling Changes

git pull origin <branch_name>

9. Viewing Commit History

Full Log

git log

Compact Log

git log --oneline

10. Undoing Changes

Unstage a File

git reset <file>

Discard Local Changes

git checkout -- <file>

Reverting a Commit

git revert <commit_id>

11. Stashing Changes

Saving Uncommitted Changes

git stash

Reapplying Stashed Changes

git stash apply

12. Tagging

Creating a Tag

git tag <tag_name>

Pushing a Tag

git push origin <tag_name

13. Additional Tips

- Use git help <command> for detailed documentation.
- Regularly pull updates in team environments to avoid conflicts:
- git pull

Git Restore, Revert, Reset Overview

1. **git restore**
    - **Purpose**: Restore files to a previous state.
    - **Use Case**: Discard local changes in a file or unstage a file.
    - **Example**:
      To discard changes in file.txt:
    - git restore file.txt
2. **git revert**
    - **Purpose**: Undo a commit by creating a new commit that reverses it.
    - **Use Case**: Revert a specific commit without removing it from history.
    - **Example**:
      To revert a commit with hash abcd1234:
    - git revert abcd1234
3. **git reset**
    - **Purpose**: Move the HEAD pointer and potentially change the working directory and staging area.
    - **Use Cases**:
        - **Soft Reset**: Keeps changes staged after moving the HEAD.
        - **Mixed Reset** (default): Moves the HEAD and updates the staging area but keeps the working directory intact.**Hard Reset**: Resets the HEAD, staging area, and working directory, discarding all changes.

- **Example (Hard Reset)**:
  To reset the repository to commit abcd1234 and discard all changes:
- git reset --hard abcd1234

THE END