**TASK 3: Build a Version-Controlled DevOps Project with Git**

**✅ Task Documentation**

**✅ STEP 1: Setup Local Project Folder**

cmd

CopyEdit

cd Desktop

mkdir devops-project

cd devops-project

📝 **Explanation**:

* cd Desktop → Changes directory to your desktop.
* mkdir devops-project → Creates a new folder named devops-project.
* cd devops-project → Enters into that new project folder.

**✅ STEP 2: Initialize Git Locally**

cmd

CopyEdit

git init

📝 **Explanation**:

* Initializes a new Git repository in the current folder.
* Creates a hidden .git folder to track changes in your files.

**✅ STEP 3: Create Initial Files**

cmd

CopyEdit

echo # DevOps Project > README.md

echo node\_modules/ > .gitignore

mkdir docs

echo # Task Documentation > docs\task-documentation.md

mkdir scripts

echo echo Hello from script > scripts\sample-script.bat

📝 **Explanation**:

* echo → Prints the given text into a file.
* > filename → Redirects the output to create or overwrite the file.
* mkdir → Makes a new folder.

**Result**:

* A README.md file with title text.
* A .gitignore file telling Git to ignore node\_modules/.
* A docs/ folder with a documentation file.
* A scripts/ folder with a sample .bat script.

**✅ STEP 4: Make Initial Commit**

cmd

CopyEdit

git add .

git commit -m "Initial commit with base files"

📝 **Explanation**:

* git add . → Stages all files in the current folder for commit.
* git commit -m "..." → Saves the changes with a message describing what you did.

**✅ STEP 5: Create GitHub Repository**

📌 **Action**:  
Go to [GitHub](https://github.com) → Click “New Repository” → Name it devops-project → Create repo (no need to initialize with README).

📝 **Explanation**:

* You're creating an empty repo on GitHub to push your local code to.

**✅ STEP 6: Link GitHub Repo to Local**

cmd

CopyEdit

git remote add origin https://github.com/your-username/devops-project.git

git branch -M main

git push -u origin main

📝 **Explanation**:

* git remote add origin URL → Links your local repo to GitHub.
* git branch -M main → Renames your branch to main.
* git push -u origin main → Pushes your code to the GitHub main branch.

**✅ STEP 7: Create Development and Feature Branches**

cmd

CopyEdit

git checkout -b dev

git push -u origin dev

git checkout -b feature/sample-feature

git push -u origin feature/sample-feature

📝 **Explanation**:

* git checkout -b branch-name → Creates and switches to a new branch.
* git push -u origin branch-name → Pushes the new branch to GitHub and sets it as the default tracking branch.

**✅ STEP 8: Make a Change in Feature Branch**

cmd

CopyEdit

notepad scripts\sample-script.bat

📝 **Explanation**:

* Opens the script file in Notepad for editing.
* You can add:

bat

CopyEdit

@echo off

echo Hello from DevOps project feature!

cmd

CopyEdit

git add scripts\sample-script.bat

git commit -m "Updated sample script in feature branch"

git push

📝 **Explanation**:

* git add → Stages your updated file.
* git commit → Saves your changes with a message.
* git push → Pushes changes to GitHub.

**✅ STEP 9: Create Pull Request (PR)**

📌 **Action**:  
On GitHub:

* Go to feature/sample-feature branch.
* Click **“Compare & Pull Request”**
* Base = dev, Compare = feature/sample-feature
* Click **“Create Pull Request”**

📝 **Explanation**:

* You're requesting to merge feature code into the dev branch.
* It allows team review before merging.

**✅ STEP 10: Merge PRs**

📌 **Action**:

* On GitHub, click **"Merge Pull Request"** on the open PR page.

📌 **Repeat**:

* Do another PR from dev → main the same way.

📝 **Explanation**:

* This merges your tested code into the next branch in line.
* Keeps the main branch clean and production-ready.

**✅ STEP 11: Add a Version Tag**

cmd

CopyEdit

git checkout main

git pull origin main

git tag -a v1.0 -m "First stable release"

git push origin v1.0

📝 **Explanation**:

* git tag -a v1.0 → Adds a version label to your commit.
* git push origin v1.0 → Pushes that tag to GitHub.

**✅ STEP 12: Add More Documentation**

cmd

CopyEdit

notepad docs\task-documentation.md

📝 **Explanation**:

* Open the documentation file in Notepad to update your tasks.

Paste the full markdown documentation you’ve built (you can use this doc itself).

cmd

CopyEdit

git add docs\task-documentation.md

git commit -m "Added task documentation"

git push

**🔄 Quick Summary Table:**

| **Flag** | **Stands for** | **Use In Command** | **Meaning** |
| --- | --- | --- | --- |
| -m | message | git commit -m / git tag -m | Write a commit or tag message directly |
| -a | annotate | git tag -a | Create a tag with full metadata (recommended for release) |
| -u | upstream | git push -u | Set the default remote branch for future pushes |

**📌 1️⃣ What is Git?**

**Git** is a distributed version control system used to track changes in source code during software development. It allows multiple developers to work on a project simultaneously, manage code history, and collaborate efficiently. Each developer has a local copy of the entire code repository, and changes can be shared or merged with others.

**📌 2️⃣ What is the difference between merge and rebase?**

* **Merge** combines the changes from one branch into another, creating a new "merge commit" to preserve the history of both branches.
* **Rebase** moves or replays a series of commits from one branch onto another, placing them as if they were created after the latest commit in the target branch — resulting in a cleaner, linear project history without merge commits.

**Key Difference:**

* *Merge preserves history with extra commits.*
* *Rebase rewrites history for a clean, linear progression.*

**📌 3️⃣ What is a Pull Request?**

A **Pull Request (PR)** is a way to propose changes you've made in your branch to be merged into a main or another branch in a collaborative project. It’s a review mechanism in platforms like GitHub, GitLab, or Bitbucket, where team members can review, discuss, and approve code before it's merged.

**📌 4️⃣ How do you resolve merge conflicts?**

A **merge conflict** happens when Git can't automatically merge changes because the same part of the file was modified in different branches.

**📌 Example Scenario:**

**Let’s say you have a file message.txt:**

**On main branch:**

**text**

**CopyEdit**

**Hello World!**

**This is the main branch.**

**On feature branch:**

**text**

**CopyEdit**

**Hello World!**

**This is the feature branch.**

**Now if you try to merge feature into main, Git won’t know which line to keep because both modified the same line.**

**📌 How to Resolve a Merge Conflict (Step-by-Step):**

**✅ 1️⃣ Run git status**

**To see which files have conflicts.**

**bash**

**CopyEdit**

**git status**

**It will show something like:**

**scss**

**CopyEdit**

**both modified: message.txt**

**✅ 2️⃣ Open the Conflicting File**

**When you open message.txt, you’ll see conflict markers like this:**

**text**

**CopyEdit**

**Hello World!**

**<<<<<<< HEAD**

**This is the main branch.**

**=======**

**This is the feature branch.**

**>>>>>>> feature**

**✅ 3️⃣ Manually Edit the File**

**Decide which version to keep — or even combine them.**

**For example:**

**text**

**CopyEdit**

**Hello World!**

**This is the updated message after merge.**

**✅ 4️⃣ Stage the Resolved File**

**Once edited, tell Git it’s resolved by adding it:**

**bash**

**CopyEdit**

**git add message.txt**

**✅ 5️⃣ Complete the Merge**

**Now finish the merge:**

**bash**

**CopyEdit**

**git commit**

**If you had started with git merge feature, this commit will finalize it after resolving conflicts.**

**📌 5️⃣ What are Git Tags?**

**Git tags** are labels or references that point to specific commits in the repository history. They are often used to mark **release points (v1.0, v2.0)** or important milestones in a project.

Two types:

* **Lightweight tag**: Simple pointer to a commit.
* **Annotated tag**: Includes metadata like the tagger’s name, date, and a message.

**📌 6️⃣ What is Git Workflow?**

A **Git workflow** defines a set of rules and branching strategies that a team follows when using Git to manage a project.

**Common workflows:**

* **Centralized Workflow**: Single main branch, everyone pushes/pulls from it.
* **Feature Branch Workflow**: Separate branches for new features.
* **Gitflow Workflow**: Structured branching with master, develop, feature, release, and hotfix branches.
* **Forking Workflow**: Developers fork the main repository, make changes, then create pull requests.

**📌 7️⃣ Explain git stash.**

git stash temporarily saves changes that are not ready to be committed, allowing you to switch branches or work on something else without losing your current modifications.

**Usage:**

* git stash — saves current changes.
* git stash list — view stashed changes.
* git stash apply — reapply the stashed changes.
* git stash drop — remove a stash.

**📌 8️⃣ What is the use of .gitignore?**

A **.gitignore** file tells Git which files or directories to ignore and not track in version control. This is useful for excluding temporary files, build artifacts, sensitive data, or files generated by your operating system or development tools.

**Example:**

bash

CopyEdit

node\_modules/

\*.log

.env

.DS\_Store