**Centralized Version Control Systems (CVCS)**

* **Single Central Repository**: One main server holds all the code.
* **Examples**: Subversion (SVN), Perforce.
* **Pros**: Easy to manage and understand, centralized control.
* **Cons**: If the server is down, no one can commit changes; requires network access.

**Distributed Version Control Systems (DVCS)**

* **Multiple Repositories**: Every developer has a full copy of the repository.
* **Examples**: Git, Mercurial.
* **Pros**: Can work offline, faster operations, better at handling branches and merges.
* **Cons**: More complex to understand, potential for diverged codebases if not managed well.

**In summary**: CVCS has a single central repo and requires network access, while DVCS allows for offline work with each developer having a full copy of the repo.

 **Local Working Directory**: Where you make changes to your files.

 **Staging Area**: Where you prepare and review changes before committing.

 **Git Repository**: Where your project's history is stored and managed.

**GIT**:

EC2—UBNTU (pre install)

git –version

git config --global user.name "Your Name"

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

make folder—mkdir mygitfolder—>git init.

git init is a command used in Git to initialize a new Git repository. When you run git init in a directory, it sets up all the necessary files and directories that Git needs to start tracking changes to files in that directory.

Create files—vi file1

git add file1

git add . 🡪 **Add all changes in the directory to the staging area**:

git status

git commit -m “my file1 created” 🡪 **Commit changes with a message**

git status

git log --oneline

git restore --staged <file> **🡪** **to unstage**

**GITHUB—create repo—proj\_devops**

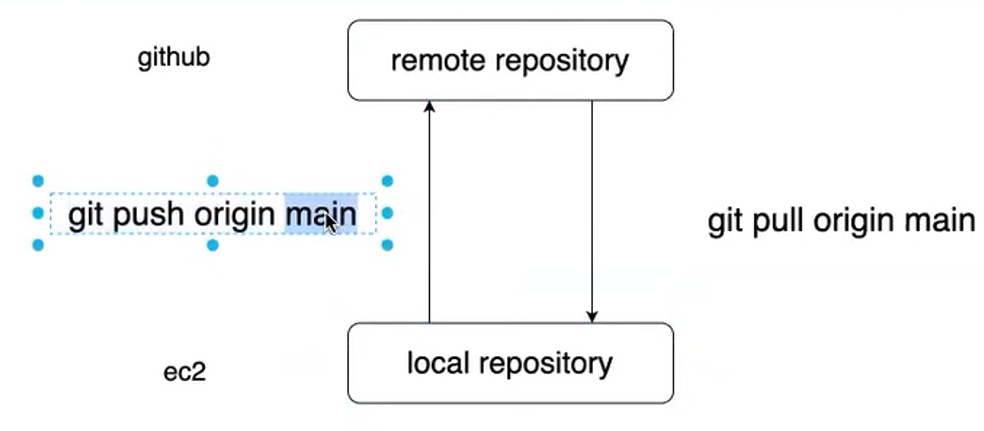
**git remote add origin** [**https://github.com/Bhavesh-55/proj\_devops.git**](https://github.com/Bhavesh-55/proj_devops.git)

The command git remote add origin <repository\_url> is used to add a remote repository to your local Git repository.

git branch

git branch -M main

The command git branch -M main is used to rename the current branch to main. This is commonly done to change the default branch name from master to main, which has become a more widely accepted practice.



git push origin main

**# Push the 'main' branch to the remote repository and set it to track the remote 'main' branch.**

Future pushes and pulls can be done with simpler commands:

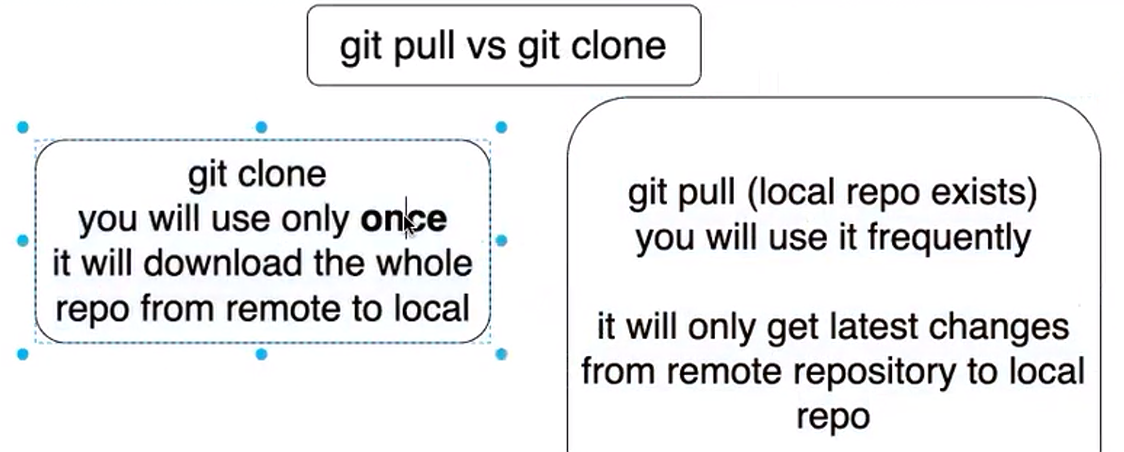
* **Push changes**: git push
* **Pull changes**: git pull-- Updates your existing local repository with the latest changes from the remote branch.

The command rm -rf \* is a powerful and potentially dangerous command that deletes all files and directories in the current directory without prompting for confirmation. Here's a breakdown of what it does:

Now git pull origin main -🡪 It won’t work as we have deleted all the files and folders/repo.

The **git clone** command is used to create a copy of an existing Git repository. This command downloads the repository from a remote location and sets it up on your local machine.

git clone <https://github.com/Bhavesh-55/proj_devops.git>



**BRANCH**:

**switch branches**:

git checkout <branch\_name>

git switch <branch\_name>

**List All Local Branches**:

git branch

**Delete a Local Branch**: (d—normal and D—force delete)

git branch -d <branch\_name>

**Rename the Current Branch**:

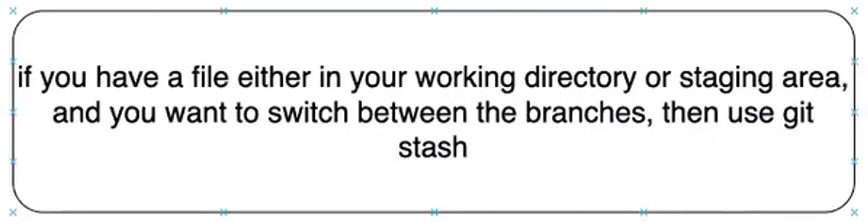
git branch -m <new\_branch\_name>

New branch created—new file1 created—It won’t reflect in main branch.

Goto main branch—merge new branch.

**Merge the Source Branch into the Current Branch**:

git merge source\_branch



**Save Changes to the Stash (to hide)**:

git stash -u --u for untrack files.

**Remove the Most Recent Stash**:

git stash pop

If work is not related to specific branch until we commit. It would be visible in the all-available branches. git stash to be used while switching.

**Stash Your Changes, Including Untracked Files**: (backup of both files. Working dir and staging area)

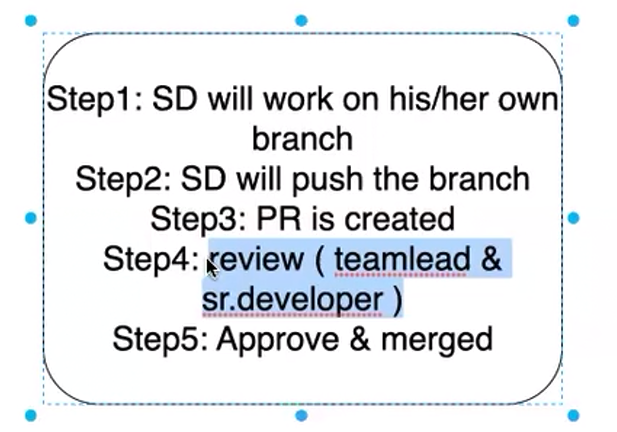
git stash -u

**Revert the Commit**: undo the changes of specific commit id and also create a commit id.

git log --oneline

git revert <commit id>

**Pull request (PR): (Merge branch with review and approval)**



Create new branch—switch to new branch—create files—add—commit—push to remote repo

git push origin <newbranch>

username & password won’t work.

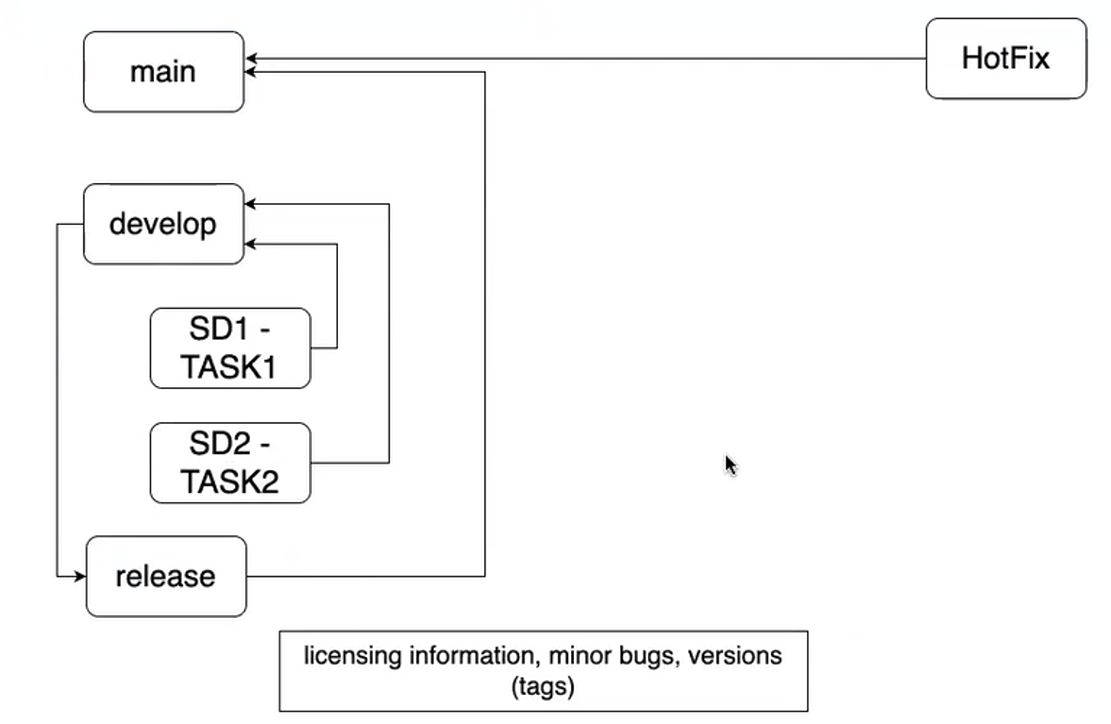
git remote set-url origin <https://Bhavesh-55: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@github.com/Bhavesh-55/proj_devops.git>

it is valid till 90 days(auth method)

git push origin <newbranch>

Goto gui—PULL REQ—NEW PR—NEWBRANCH—CREATE PR—msg—create PR—reviewer—merge.

**Gitflow workflow** is a Git **branching model** that defines a strict branching structure for managing the development process.



**feature Branch**:

Used to develop new features. Branched off from develop and merged back into develop.

**Develop Branch: It** reflects latest delivered development changes for the next release.

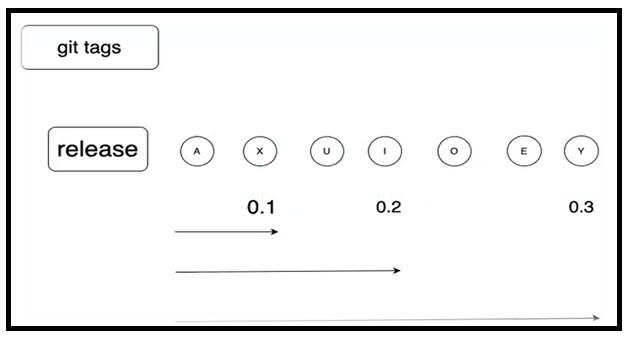
**Release**:

It allows for finalizing the release by performing last-minute fixes and **versioning/tag** before merging the changes into the main and develop branches.

**hotfix**:

* For quick fixes to production.
* Branched off from main and merged back into both main and develop.

GIT TAG: **Tags mark specific points in history**, often releases.



git log –oneline

**Create tag:**

git tag < tag name > : (latest commit by default)

git tag <tag name> <commit id> :EX. git tag 1.1 bb549h74

git log –oneline

**Delete tag:**

git tag -d < tag name >

**List tags**:

git tag

**Now Push the release branch to remote repo:**

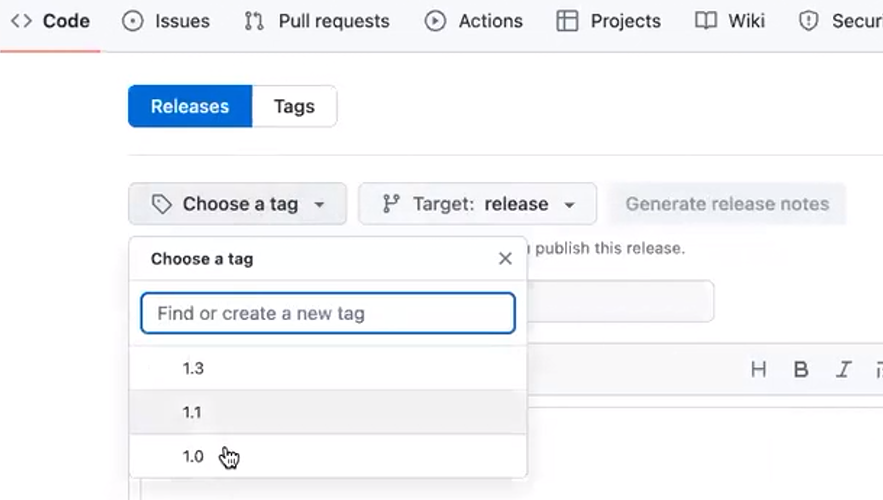
git branch

git push origin release

**Push tag:**

git push origin <tag name>

git push origin –tags

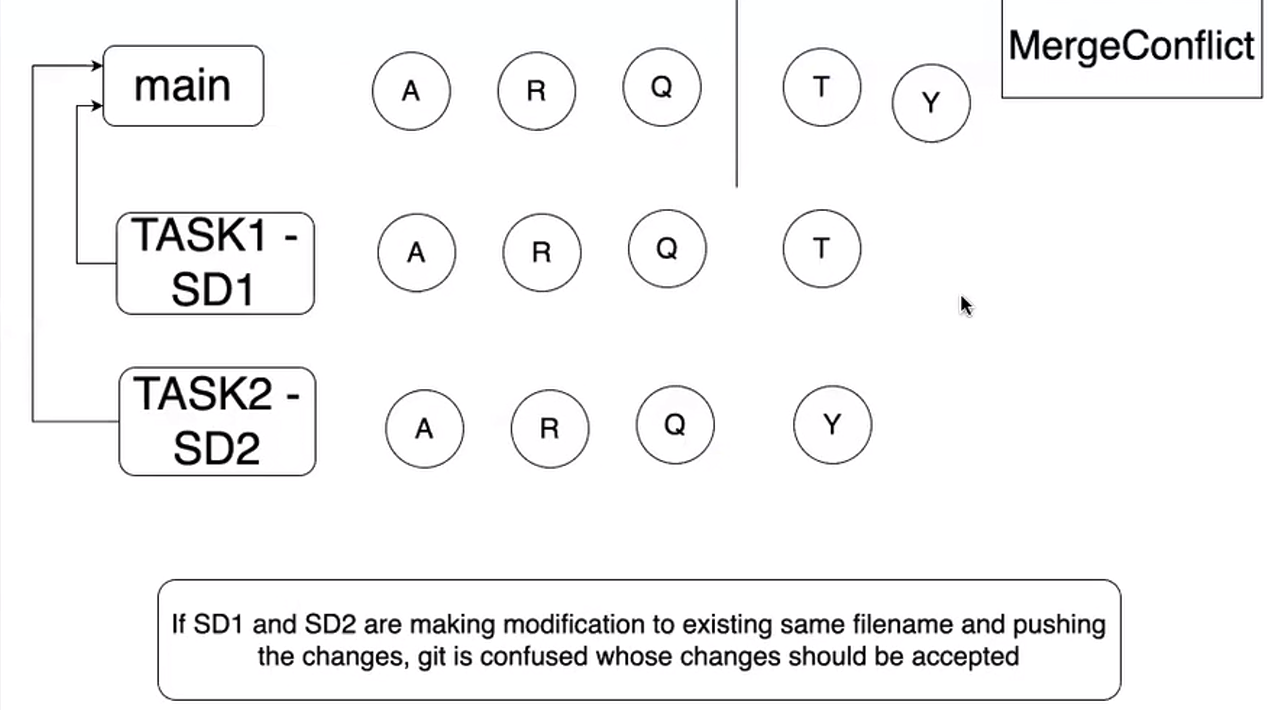


GUI—tag—release—select branch and tag—description can be done with TAG.

**Merge Conflict in Git:**

Happens when changes in different branches (same file) can't be automatically merged by Git.

When one person has deleted a file while another person has modified it.

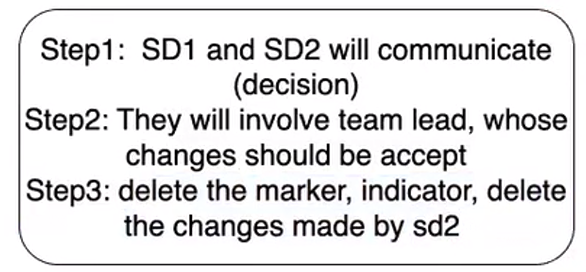


Task1 branch –push to remote repo—review-accept and merged.

Task2 branch –push to remote repo—auto conflict occur—

Now 3 options—only task 1 accept or only task 2 accept or both can be accepted.

**Conflict Markers**: <<<<<<, ======, >>>>>> indicate where conflicts occur.



**Resolve Conflicts**: Manually edit the files, stage the changes, and commit the merge.

**My task:**

**FIRST CLONE THE REPO:**

Main branch: file1, f2, f3

Create newbranch\_1----by default all file will be present

Create newbranch\_2—same

Now change file2(branch2)—add --commit

Now change file2(branch1)—add –commit

git push origin newbranch1 – PR—ACCEPT-MERGE

git push origin newbranch2—PR—check & modify—remove markers and merge.

**Fork in Git:**

**Key Points**

1. **Create a Copy**: Forking creates a personal copy of someone else's repository on your GitHub account.
2. **Independent Changes**: You can make changes, add features, and experiment in your forked repository without affecting the original project.
3. **Collaborate**: When you’re ready to share your changes, you can submit a pull request to the original repository to have your changes reviewed and potentially merged.

**Git Merge:**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Git Merge** | **Git Rebase** |
| **Commit History** | Creates a merge commit id, resulting in a non-linear history | Rewrites commit history, resulting in a **linear history** |
| **Use Case** | Suitable for preserving the full history of both branches in collaborative work | Ideal for maintaining a clean, linear commit history for local changes |

**Git Commands:**

## 1. Initialize a local repository

## 2. Add a file to the staging area



To add all files in the current directory, use . in place of <file>.



## 3. Check the status of the repository



## 4. Commit changes



## 5. Add a remote repository



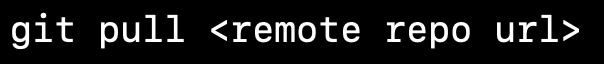
## 6. Push changes to a remote repository branch

## 

## 7. Clone a remote repository



## 8. Pull changes from a remote repository



## 9. Remove a file from the staging area

## 

## 10. Remove a file from the repository



Note: After doing git rm <file>, we also have to do git commit -m "message"

## 11. Create a branch



  You can **create** a **branch** and **switch** to it using the checkout   command.



## 12. Switch to a branch

## 13. Delete a branch



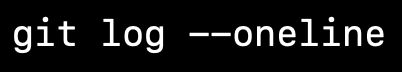
## 14. List branches



## 15. Display the commit history



## 16. Display the commit history in online



## 17. Stash changes : **The stash allows you to temporarily store changes without committing them.**

## **To stash your staged files without committing them**



To stash your staged files along with untracked files then



## 18. List stashes



## 19. Restore changes from stash and remove it



## 20. Revert a commit

This command helps you to reverting a commit to previous version



## 21. Display the changes between two commits



## 22. Merge a branch



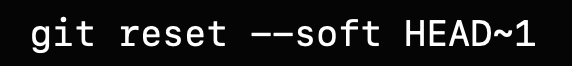
## 23. rebase a branch



## 24. hard Reset



## 25. soft Reset



## 26. mixed Reset

