## DevOps Day 2 **1. Introduction to SCM Tool (Source Code Management)**

* SCM stands for **Source Code Management**.
* It helps developers **track, manage, and control code changes** during software development.
* It allows multiple people to work together on the same project safely.
* SCM keeps a **version history**, so you can easily go back to older code.

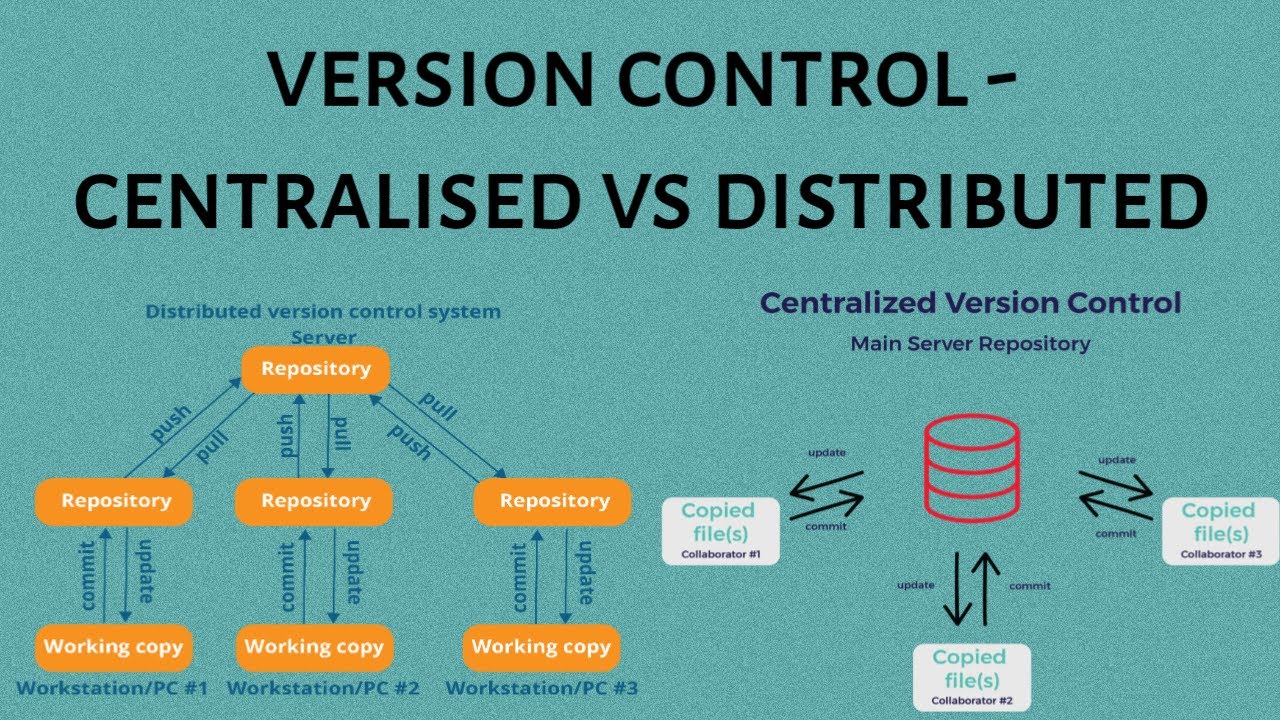
**Examples:** Git, SVN (Subversion), Mercurial, Perforce.

**Key Benefits:**

* Tracks every code change.
* Helps teamwork and collaboration.
* Prevents code loss or overwriting.
* Allows rollback to earlier versions.

## **2. Difference Between Centralized and Distributed VCS**

| **Aspect** | **Centralized VCS (CVCS)** | **Distributed VCS (DVCS)** |
| --- | --- | --- |
| **Repository** | One central server stores all code. | Every developer has a full local copy of the repository. |
| **Internet Requirement** | Needs internet to commit or update code. | Can work offline and sync later. |
| **Speed** | Slower – depends on network and central server. | Faster – most work happens locally. |
| **Data Safety** | If server crashes, data may be lost. | Every user has a backup copy. |
| **Examples** | SVN, CVS | Git, Mercurial |



## **3. Introduction to Git**

* **Git** is a **Distributed Version Control System (DVCS)** created by **Linus Torvalds** in 2005.
* It helps teams manage source code, track changes, and work in parallel.

**Key Features:**

* Works offline (local commits).
* Fast and reliable.
* Supports branching and merging.
* Keeps full project history.
* Used with hosting services like **GitHub**, **GitLab**, and **Bitbucket**.

## **Workflow Of GIT** **4. Git Branches and Common Operations**

### **Git Branch**

* A **branch** in Git is like a separate line of development.
* It lets you work on a new feature or fix without affecting the main code.

**Common Branch Types:**

* **Main / Master:** The main production-ready branch.
* **Develop:** For ongoing development.
* **Feature:** For new features or modules.
* **Bugfix:** For fixing specific issues.
* **Release:** For preparing final versions before deployment.

**Common Commands:**

* git branch → Lists all branches.
* git branch <branch-name> → Creates a new branch.
* git checkout <branch-name> → Switches to another branch.
* git merge <branch-name> → Combines another branch into the current branch.
* git diff → Shows the difference between file versions or branches.

## **5. Introduction to Remote Repository (GitHub)**

* A **Remote Repository** is an online version of your Git project.
* It allows team members to **share, sync, and collaborate** from different systems.
* **GitHub** is the most popular remote repository hosting platform.

**Common Git Remote Commands:**

* git remote add origin <repo-url> → Connect your local repo to a remote one.
* git push origin <branch-name> → Upload local changes to GitHub.
* git pull origin <branch-name> → Download and merge changes from GitHub.
* git clone <repo-url> → Copy a remote repository to your local system.

**Benefits of Remote Repositories:**

* Enables teamwork and collaboration.
* Keeps backups of your code online.
* Integrates with CI/CD tools for automation.

# Helps with version tracking and open-source sharing. GIT Commands (Work on Local Repository)

## git init

Create a new empty Git repository in your current folder.  
  
Use:  
 git init  
  
Example:  
 If you have a folder named 'myproject', open it in terminal and run 'git init' to start version control.

## git add

Add files to the staging area (prepare them for commit).  
  
Use:  
 git add <filename>  
 git add . # adds all files in the folder  
  
Example:  
 git add index.html

## git commit

Save changes from the staging area to the local repository with a message.  
  
Use:  
 git commit -m "Your commit message"  
  
Example:  
 git commit -m "Added homepage file"

## git status

Show the current state of files — which are modified, staged, or untracked.  
  
Use:  
 git status  
  
Example Output:  
 modified: index.html  
 untracked files: style.css

## git log

View the history of commits in the repository.  
  
Use:  
 git log  
  
Example Output:  
 commit a1b2c3d4  
 Author: Shivansh  
 Date: Thu Oct 09 17:00 2025  
 Message: Added homepage

## git restore

Undo changes in files that are not yet staged or committed.  
  
Use:  
 git restore <filename>  
  
Example:  
 git restore index.html  
 (Restores the file to the last committed version)

## git revert

Undo a specific commit by creating a new commit that reverses it (keeps history safe).  
  
Use:  
 git revert <commit\_id>  
  
Example:  
 git revert a1b2c3d4  
 (This creates a new commit that cancels out the changes of the mentioned commit.)

## Summary Table

|  |  |  |
| --- | --- | --- |
| Command | Description | Example |
| git init | Start a new Git repository | git init |
| git add | Stage files for commit | git add . |
| git commit | Save changes with message | git commit -m "msg" |
| git status | Check current status | git status |
| git log | View commit history | git log |
| git restore | Undo unstaged changes | git restore file.txt |
| git revert | Undo a specific commit | git revert id |

# Exercise 1.1: Create and Initialize a Repository

mkdir my-project

cd my-project

git init

# Exercise 1.2: Verify initialization

ls -la # Linux/Mac

dir /a # Windows

# You should see a .git folder

# Exercise 1.3: Check git status in empty repo

git status

-----------------------------------------------------------------------  
  
  
  
  
  
# Exercise 2.1: Check initial status

git status

# Exercise 2.2: Create files and check status

echo "Hello Git" > file1.txt

echo "Learning Git" > file2.txt

git status

# Notice: Files appear as "untracked"

# Exercise 2.3: Create multiple files and check

mkdir docs

echo "Documentation" > docs/readme.txt

git status

# Notice: Entire folder appears as untracked  
  
  
  
  
------------------------------------------------------------------------------------

# Exercise 3.1: Add single file

git add file1.txt

git status

# Notice: file1.txt is now "staged" (green), file2.txt still "untracked" (red)

# Exercise 3.2: Add multiple files

git add file2.txt docs/readme.txt

git status

# All files should now be staged

# Exercise 3.3: Modify and stage

echo "Additional line" >> file1.txt

git status

# Notice: file1.txt appears in both staged and modified

# Exercise 3.4: Stage all changes

git add .

git status

# Exercise 4.1: First commit

git commit -m "Initial commit: Added project files"

git status

# Working directory should be clean

# Exercise 4.2: Make changes and commit

echo "New feature" > feature.txt

git add feature.txt

git commit -m "Add feature file"

# Exercise 4.3: Modify and commit with combined command

echo "Update" >> file1.txt

git commit -am "Update file1 content"

# Note: This only works for already tracked files

# Exercise 4.4: Multi-line commit message

git add .

git commit -m "Major update

- Added new documentation

- Updated existing files

- Prepared for next phase"

# Exercise 5.1: Basic log

git log

# Exercise 5.2: Compact log view

git log --oneline

# Exercise 5.3: Limited commits

git log -3 # Last 3 commits

git log --oneline -5 # Last 5 commits in one line

# Exercise 5.4: Detailed log with statistics

git log --stat

# Exercise 5.5: Custom format

git log --pretty=format:"%h - %an, %ar : %s"

# %h = hash, %an = author, %ar = relative time, %s = subject  
  
# Exercise 6.1: Discard working directory changes

echo "Unwanted change" >> file1.txt

git status

git restore file1.txt

git status

cat file1.txt # Change is gone

# Exercise 6.2: Restore multiple files

echo "Bad edit" >> file1.txt

echo "Another bad edit" >> file2.txt

git restore . # Restore all files

git status

# Exercise 6.3: Unstage files

echo "New content" > new-file.txt

git add new-file.txt

git status

git restore --staged new-file.txt

git status # File is unstaged but still exists

# Exercise 6.4: Restore to specific commit

git log --oneline

git restore --source=HEAD~1 file1.txt # Restore from previous commit  
  
# Exercise 7.1: Setup - Create commits to revert

echo "Version 1" > app.txt

git add app.txt

git commit -m "Add app.txt version 1"

echo "Version 2" >> app.txt

git commit -am "Update to version 2"

echo "Version 3 - Bug!" >> app.txt

git commit -am "Update to version 3 with bug"

git log --oneline

# Exercise 7.2: Revert the last commit

git revert HEAD

# Editor opens for commit message, save and exit

git log --oneline # New revert commit appears

cat app.txt # Bug is removed

# Exercise 7.3: Revert a specific commit

git log --oneline

git revert HEAD~1 # Revert second-to-last commit

# Might cause conflicts - resolve if needed

# Exercise 7.4: Revert without auto-commit

echo "Feature A" > feature-a.txt

git add feature-a.txt

git commit -m "Add feature A"

git revert HEAD --no-commit

git status # Changes staged but not committed

git commit -m "Manually revert feature A"

## 

## 📝 Comprehensive Practice Project

### Create a Mini Todo List Project

# Step 1: Initialize project

mkdir todo-app

cd todo-app

git init

# Step 2: Create initial files

echo "# Todo List App" > README.md

echo "Buy groceries" > tasks.txt

git status

# Step 3: Stage and commit

git add .

git commit -m "Initial commit: Setup todo app"

# Step 4: Add more tasks

echo "Complete Git tutorial" >> tasks.txt

echo "Review notes" >> tasks.txt

git status

git diff # See what changed

git add tasks.txt

git commit -m "Add learning tasks"

# Step 5: Create mistake and fix it

echo "This is wrong!" > tasks.txt

git status

git restore tasks.txt # Undo the mistake

# Step 6: Add priority system

echo "HIGH: Fix bugs" > priority-tasks.txt

echo "LOW: Clean desk" >> priority-tasks.txt

git add priority-tasks.txt

git commit -m "Add priority task system"

# Step 7: View history

git log --oneline --graph

# Step 8: Accidentally delete content

echo "" > README.md

git add README.md

git commit -m "Accidentally cleared README"

# Step 9: Revert the mistake

git log --oneline

git revert HEAD

git log --oneline

# Step 10: Final status check

git status

ls

cat README.md # Should be restored