Git and Version Control

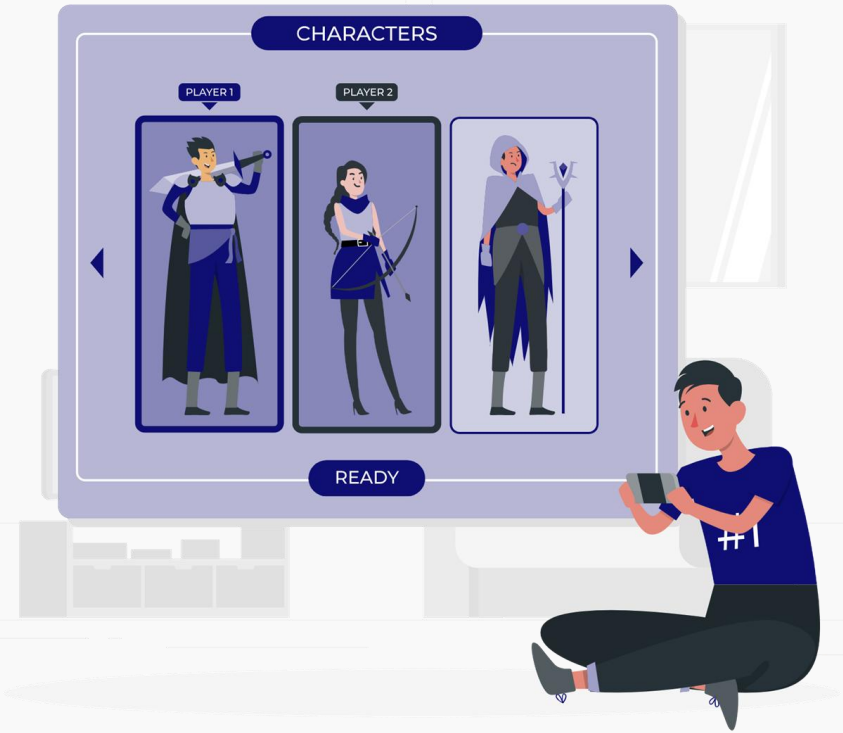
Week 01 | Session 01

konvecta

Learning Objectives

By the end of this session, students will be able to:

- Clone, commit, and push changes using Git commands.
- create branches and merge changes effectively.



Ice Breaker

konvecta

Introduction to Version Control Systems (VCS)

What is a Version Control System (VCS)?

- Software that helps a team of software developers work together.
- Tracks and manages changes to files over time.
- Allows you to revert files to a previous state, compare changes, see who modified what, and recover lost work.

Introduction to Version Control Systems (VCS)

Centralized VCS (CVCS) vs. Distributed VCS (DVCS):

CVCS (e.g., SVN, Perforce): Single central server holds all versions of the project. Developers check out files, make changes, and check them back in.

- *Pros*: Simpler to set up initially.
- *Cons*: Single point of failure; requires network access; slower for large teams.

DVCS (e.g., Git, Mercurial): Every developer has a complete copy (clone) of the entire repository, including its full history.

- *Pros*: No single point of failure; faster operations (most are local); offline work possible; excellent for branching and merging.
- *Cons*: Slightly steeper initial learning curve.

Git is a DVCS: This means you have a full copy of the project history on your local machine.

Basic Workflow in Git

Working Directory:

- The actual files you see and edit on your computer.
- These are the 'untracked' or 'modified' files.

Staging Area (Index):

- A temporary area where you prepare changes before committing them.
- You add specific changes (files or parts of files) that you want to include in your next commit.
- Think of it as a 'holding area' for what you're about to save.

Local Repository:

- The .git directory on your local machine.
- Contains all the committed changes (your project's history).
- This is your personal copy of the entire project history.

Remote Repository:

- A version of your repository hosted on a server (e.g., GitHub, GitLab, Bitbucket).
- Used for sharing changes with collaborators and as a central backup.

Git Commands: The Essentials

git init: Initializes a new, empty Git repository in the current directory. (Only done once per project).

git clone [URL]: Creates a copy of an existing remote repository on your local machine.

git status: Shows the status of your working directory and staging area (which files are modified, staged, or untracked).

git add [file(s)]: Adds changes from the working directory to the staging area.

git add .: Adds all new and modified files in the current directory and subdirectories.

git commit -m "Your descriptive message": Records the staged changes permanently into the local repository.

The message is crucial for understanding the commit's purpose.

git push [remote] [branch]: Uploads your local commits to the specified remote repository and branch.

git push origin main: Pushes to the main branch on the origin remote.

git pull [remote] [branch]: Fetches changes from the remote repository and merges them into your current local branch.

git pull origin main: Pulls from the main branch on the origin remote.

Branching and Merging: Collaborative Development

Why Branch?

- Allows developers to work on different features or bug fixes in parallel without interfering with each other's work.
- Isolates changes until they are ready to be integrated into the main codebase.

Creating Branches:

- git branch [branch-name]: Creates a new branch (but doesn't switch to it).
- git checkout [branch-name]: Switches to an existing branch.
- git checkout -b [new-branch-name]: Creates a new branch AND switches to it (common shortcut).

Merging Strategies:

- **Fast-Forward Merge:** Occurs when there are no new commits on the target branch (e.g., main) since your feature branch diverged. Git simply moves the pointer forward.
- **Three-Way Merge:** Occurs when both branches have diverged (new commits on both sides). Git creates a new "merge commit" to combine the histories.

Handling Merge Conflicts:

- Occurs when Git cannot automatically combine changes (e.g., the same line of code was modified differently in two branches).
- Git marks the conflicted areas in the file.
- You manually resolve the conflict by editing the file, then git add and git commit the resolution.

Best Practices & Conventions

Commit Messages:

- **Clear and Concise:** Summarize *what* changed and *why*.
- **Imperative Mood:** "Fix bug" not "Fixed bug".
- **First line (subject) < 50-72 chars:** Followed by a blank line, then detailed body (optional).
- *Example:* feat: Add user authentication module (Subject) This commit introduces a new user authentication system. (Body) It includes signup, login, and logout functionalities, and integrates with the existing user database.

Best Practices & Conventions

Branching Strategies (Conceptual):

- **Feature Branches:** Create a new branch for every new feature or bug fix. (Most common for individual tasks).
- **Git Flow:** A more complex, strict branching model for larger teams, involving main, develop, feature, release, and hotfix branches. (Good to be aware of).
- **Trunk-Based Development:** All developers commit directly to a single main branch, often with very small, frequent commits and robust testing. (Popular in CI/CD environments).

Commit Frequently, Push Regularly:

- Small, focused commits are easier to review and revert.
- Pushing regularly ensures your work is backed up and shared.

Pull Before You Push: Always git pull before git push to integrate others' changes and avoid conflicts.

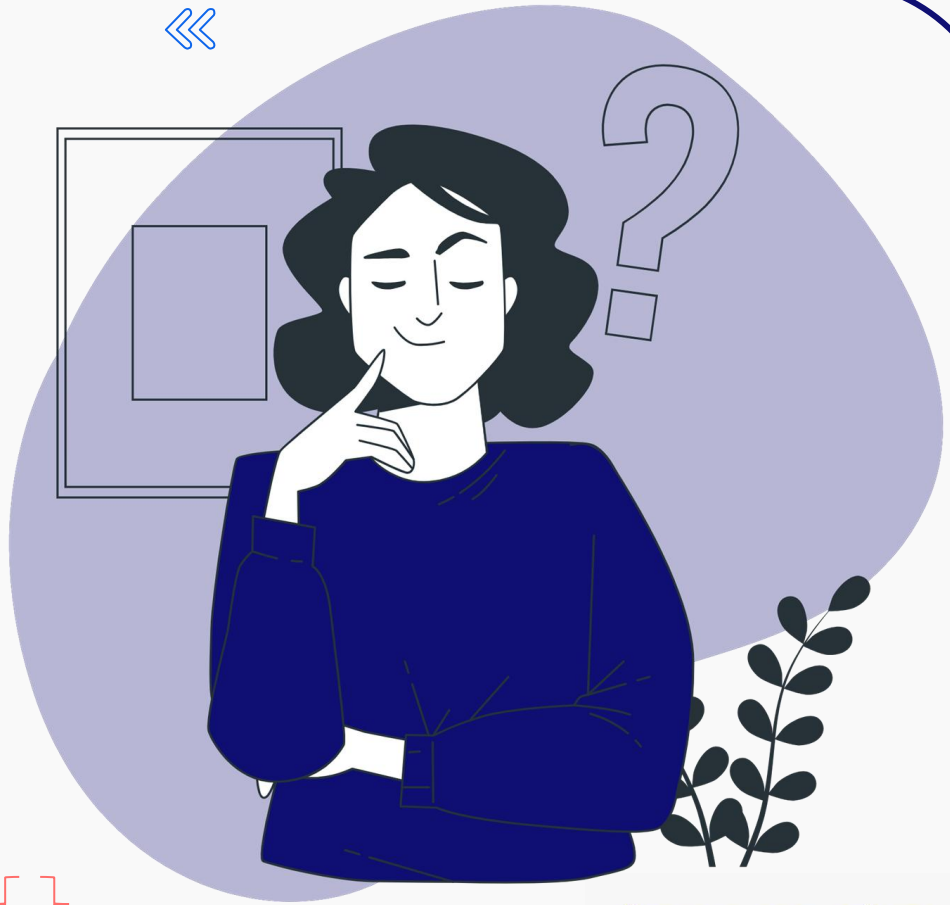
Activity

konvecta

Online Activity

konvecta

Q&A and Next Steps



konvecta

Thank You

konvecta