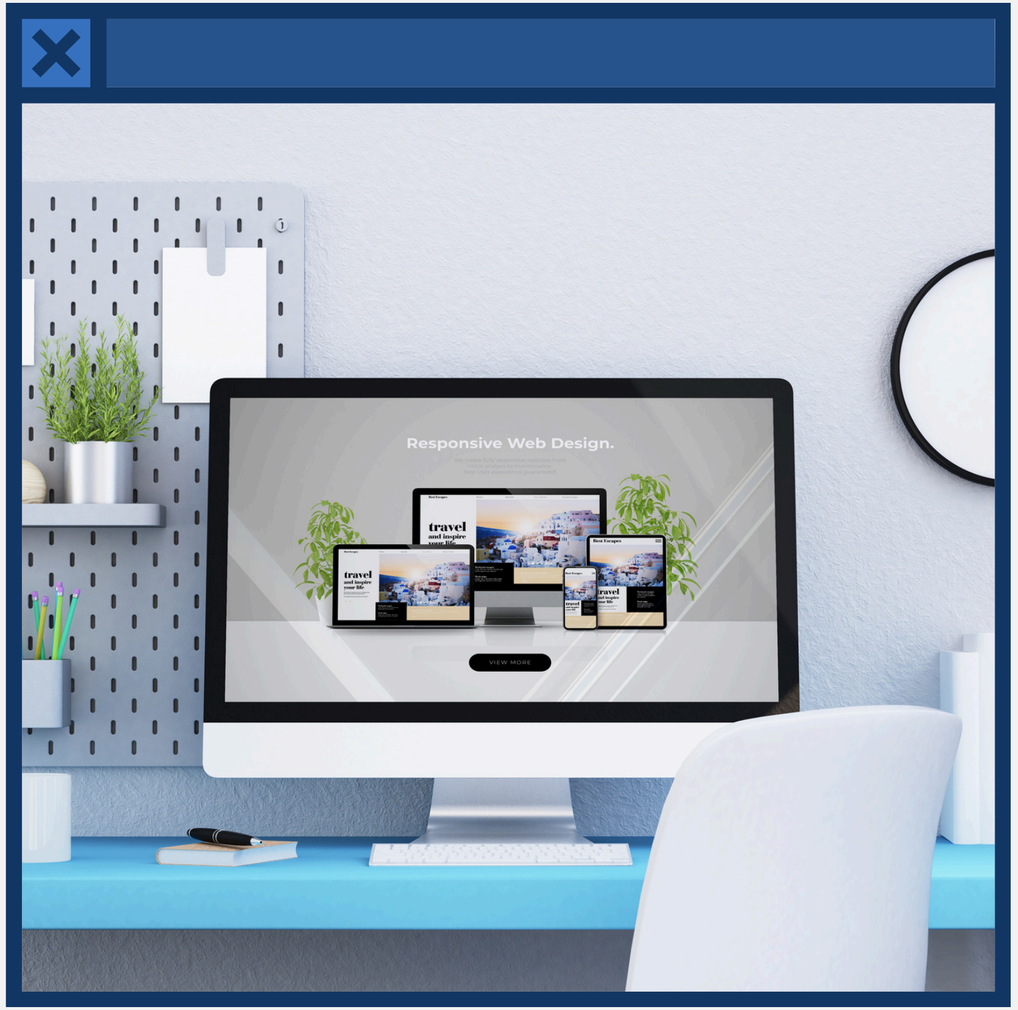
INTRODUCTION TO VERSION CONTROL



OBJECTIVES

By the end of this chapter, students should be able to:

1. Explain the purpose and benefits of version control systems in software development and project management.

1. Differentiate between local, centralized, and distributed version control systems, outlining the advantages and disadvantages of each type.

1. Apply basic Git concepts and commands to effectively manage code changes and collaborate on projects.

INTRODUCTION TO VERSION CONTROL

What is Version Control?

Version control is a system designed to track and manage changes to files over time. It allows you to save "snapshots" of your work, enabling you to revert to previous states if necessary.

Why is it Important?

- Facilitates collaboration among multiple developers.
- Records who made changes, what those changes were, and when they occurred.
- Offers a history of the project, aiding in understanding its development.
- Work on new features or fixes in isolated branches before integrating into main codebase.
- Protects against data loss from accidental deletions or errors.

TYPES OF VERSION CONTROL SYSTEMS

1. Local Version Control:

- Stores changes locally, usually as simple copies or snapshots.
- Example: Basic file copies, limited scalability.

2. Centralized Version Control (CVCS):

- Single central server holds the complete project history.
- Example: CVS, Subversion (SVN).
- Pros: Easy to manage, centralized control.
- Cons: Dependency on network access; less flexible.

TYPES OF VERSION CONTROL SYSTEMS (CONT’)

3. Distributed Version Control (DVCS):

- Every user has a complete copy of the repository, including full history.
- Example: Git, Mercurial.
- Pros: Offline work, faster operations, robust branching, powerful merging.
- Cons: Slightly more complex setup.

** In this lecture, focus primarily on Git and its advantages as a DVCS.

GIT

What is Git?

Git is a free, open-source, distributed version control system developed by Linus Torvalds. It has become the most widely used VCS globally.

Why Git?

- Provides fast and efficient management in large projects.
- Supports branching for diverse workflows.
- Facilitates teamwork through remote repositories like GitHub, GitLab, and Bitbucket.
- Offers a vast ecosystem and numerous resources.

GIT (CONT’)

Key Concepts:

- **Repository:** The database of all your project files and their history.
- **Commit:** A snapshot of your files and changes at a specific point.
- **Branch:** A separate line of development, allowing work on multiple features simultaneously.
- **Merge:** Combining changes from different branches into one.
- **Clone:** Creating a local copy of a remote repository.

BASIC GIT WORKFLOW

1. Initialize a repository (git init)
2. Add files to staging area: (git add filename)
3. Commit changes: (git commit -m "Description")
4. Create new branch: (git branch feature-x)
5. Switch to branch: (git checkout feature-x)
6. Make changes and commit in branch
7. Merge branch back to main: (git merge feature-x)
8. Push changes to remote repo: (git push origin main or branch name)
9. Pull updates from remote repo: (git pull)

This workflow supports iterative development, code review, and collaborative projects.

COMMON GIT COMMANDS

Command	Description	Example Usage
git init	Creates a new local repository	git init
git clone [url]	Clones an existing remote repository	git clone https://github.com/user/repo.git
git add [file]	Stages file(s) for next commit	git add index.html
git commit -m "message"	Records snapshot with message	git commit -m "Add homepage"
git push	Uploads local commits to remote server	git push origin main
git pull	Fetches and merges remote changes	git pull origin main
git branch [name]	Creates a new branch for parallel development	git branch feature-login
git checkout [branch]	Switches to the specified branch	git checkout feature-login
git merge [branch]	Merges specified branch into current branch	git merge feature-login
git status	Shows current branch status and uncommitted changes	git status
git log	Displays the commit history	git log

BEST PRACTICES IN VERSION CONTROL

Best Practices in Version Control:

- **Commit Frequently:**
  - Perform small, incremental commits that capture a specific change or feature. This enhances traceability and makes troubleshooting easier.
- **Write Clear Commit Messages:**
  - Clearly explain what was done and why—e.g., "Fix navbar alignment issue on mobile" or "Add login validation."
- **Use Branches for Features and Bug Fixes:**
  - Keep work in progress separate. The main or master branch should remain stable.

BEST PRACTICES IN VERSION CONTROL (CONT’)

Best Practices in Version Control:

- **Code Reviews & Pull Requests:**
  - Utilize Pull Requests (PRs) to review code changes before integrating them into the main branch. This fosters collaboration and ensures high code quality.
- **Regularly Pull Updates:**
  - Synchronize your local repository frequently with remote changes to minimize conflicts during merges.
- **Keep the Repository Clean:**
  - Remove outdated branches and refrain from committing large binary files. Implement .gitignore rules to maintain a tidy repository.

TOOLS AND PLATFORMS

- **Command Line Interface (CLI):**
  - Git Bash (Windows), Terminal (Linux/Mac) for direct command execution.

- **Graphical User Interface (GUI) Tools:**
  - Sourcetree: Visual client for Git and Mercurial.
  - GitKraken: Modern, intuitive Git client.
  - GitHub Desktop: Simplifies GitHub workflows.

TOOLS AND PLATFORMS (CONT’)

- **Hosting Platforms:**
  - GitHub: Largest community, open-source projects, integrations.
  - GitLab: Built-in CI/CD, private repositories.
  - Bitbucket: Integrates with Jira and Atlassian ecosystem.

- Example Platform:

```
git clone https://github.com/yourusername/yourproject.git
```