NUS National University of Singapore | Advanced Computing for Executives
School of Computing

Module 2: Introduction to Git

Overview of Version Control System

- Version control systematically tracks file changes over time, focusing on the source code for software and AI development. VCS tools are more efficient than manual backups.
- Version control is crucial for managing numerous source files and coordinating multiple developers in complex AI projects.
- Local VCS tracks changes using a database on a developer's laptop but lacks collaboration features and risks a single point of failure.
- Centralised VCS stores files on a central server, enabling collaboration and simplified management but still faces server downtime and failure risks.
- Distributed VCS copies a full repository to developers' computers, solving failure risks and allowing offline development. Tools like Git and Mercurial are now the standard.

Introduction to Git and GitHub

- Git, a leading distributed version control system, efficiently manages large-scale projects like Linux with over 30 million lines of code.
- Git is fast, avoids redundant data, supports branching for non-linear development, and scales well for large teams.
- Unlike traditional VCS, which uses delta-based version control, Git stores data as snapshots, ensuring faster access and better reliability.
- Git operates locally, enabling offline work, with interactions with remote repositories needed only for sharing or fetching changes.
- Git ensures data integrity with cryptographic checksums and maintains a robust, undo-friendly repository structure, minimising data loss risks.

Overview of a Git Repository

- A Git repository stores project files in three sections: the working directory, staging area, and .git directory.
- Files are either tracked (already committed or added) or untracked (newly created and not yet added).
- Tracked files can be modified, staged for the next commit, or committed and stored in the .git directory.
- The Git workflow involves modifying files, staging changes, and committing them to the .git directory.
- A Git repository can be created using git init for a new repository or git clone to copy a remote repository.

Tracking Changes to a Git Repository

- Use git init to create a new repository or git clone to copy an existing one. This sets up a working copy for adding, modifying, or deleting files.

- Stage new or modified files with git add, and commit them with git commit. Use options like -a for auto-staging changes or -m to add a commit message. Commits create snapshots traceable via unique hashsums.
- Use git rm to delete files completely, and git mv to move or rename files. Git tracks movements as deletions and new additions, requiring explicit staging for the next commit.
- Specify files to ignore in a .gitignore file using glob patterns. Ignored files, like logs or build outputs, are excluded from staging and tracking.
- Amend commits with git commit --amend, unstage files with git reset HEAD, or revert modified files to their last committed state using git checkout --. (Note that uncommitted changes may be lost permanently.)

Branching in Git

- In Git, commits link in a linear fashion, forming the main line of development.
- Branching allows tasks like error fixes or new features without affecting the main line, ensuring it stays deployment-ready.
- Branches can be merged into the main line after testing, combining changes for production.
- Git's lightweight branching is fast and efficient, encouraging frequent branching and merging for better code quality.
- Git uses branch pointers and a HEAD pointer to track branches and the current working branch.

Basic Branching and Merging Operations

- A Git repository contains a default branch, typically named main. New commits update the branch pointer to the latest commit, ensuring access to the most recent project files.
- The git branch command creates a new branch pointing to the current commit but does not switch to it. The HEAD pointer and working directory remain unchanged.
- The git checkout command switches branches, moving the HEAD pointer and updating the working directory if the branches diverge. Commits in the new branch update its pointer, leaving the original branch unchanged.
- Merging branches requires switching to the target branch and using git merge. A fast-forward merge occurs if branches align linearly, while a three-way merge resolves divergences by creating a new commit.
- The git branch -d command deletes merged branches, and git branch --no-merged lists unmerged branches. Merge conflicts, resolved manually, will be covered in a later session.

Merge Conflict Resolution

- Git has two merge types—fast-forward merges (linear commits, no conflict) and three-way merges (divergent commits, possible conflicts).
- Conflicts arise when different changes overlap; Git pauses the merge for manual resolution.
- Use git status to identify conflicts, remove conflict markers, and reconcile changes manually or with a graphical tool.
- After resolving conflicts, stage files with git add and finalise the merge with git commit.

- Minimise conflicts by modularising code and assigning file responsibilities to specific team members.

Working with a Remote Repository in GitHub

- Git's distributed nature enables remote collaboration; a remote repository must be added to your Git repository.
- Use git remote add with a short name and URL to link a remote repository; the default name is origin.
- Cloning a repository automatically sets the origin remote; use git remote to view configured remotes.
- Collaborate by pushing changes with git push and pulling or fetching changes; the initial repository upload enables sharing.
- Pull integrates changes directly; fetch downloads changes without integration, offering flexibility for managing branches.

THE END

My notes...