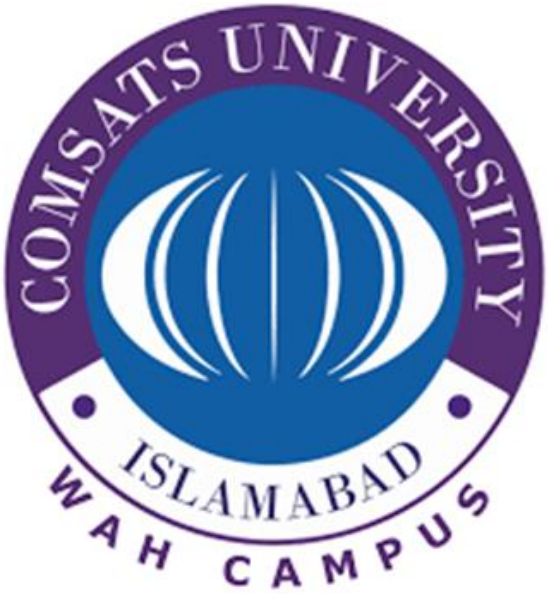
COMSATS UNIVERSITY ISLAMABAD

WAH CAMPUS



SOFTWARE QUALITY ENGINEERING

REPORT ON VERSION CONTROL TOOLS

Submitted to:

Dr. Wasif Nisar

Submitted by:

Syed M. Hamza Hassan

Reg. no:

SP23-BSE- 016

Date: 25/11/25

DEPRTMENT OF COMPUTER SCIENCE

Contents

1. Executive Summary 3

2. The Distinction: SCM vs. VCS 3

3. Categories of Version Control Tools 3

4. Key Version Control Tools 3

A. Git 3

B. Apache Subversion (SVN) 4

C. Perforce (Helix Core) 4

D. Mercurial 4

5. Functionality with Respect to Configuration Management 4

1. Configuration Identification (Baselines) 5

2. Change Control & Auditing 5

3. Infrastructure as Code (IaC) integration 5

Conclusion 5

1. Executive Summary

This report analyzes the landscape of Version Control Systems (VCS) and their critical function within the broader discipline of **Software Configuration Management (SCM)**. While SCM encompasses the entire process of managing changes to software and infrastructure, Version Control provides the technical mechanism to record, track, and merge these changes.

Modern software delivery relies on treating "everything as code"—including infrastructure, network settings, and application logic. Consequently, the role of VCS has expanded from simple code backup to becoming the "Single Source of Truth" for the entire IT environment.

2. The Distinction: SCM vs. VCS

It is vital to distinguish between the discipline and the tool:

- **Software Configuration Management (SCM):** The *process* of controlling, identifying, and auditing changes to configuration items (code, documents, settings) throughout the system lifecycle.
- **Version Control System (VCS):** The *tool* used to implement SCM strategies. It automates the recording of history and the retrieval of specific versions.

3. Categories of Version Control Tools

Feature	Centralized VCS (CVCS)	Distributed VCS (DVCS)
Architecture	A single central server contains the full version history. Users check out a working copy.	Every user has a full local copy of the entire repository history.
Dependency	Requires network connection for most operations (history, committing).	Most operations (commit, diff, log) are local and offline.
Collaboration	"Lock-Modify-Unlock" or "Copy-Modify-Merge". High risk of conflict if locks aren't used.	"Clone-Modify-Push". Encourages branching and merging.
Examples	Subversion (SVN), Perforce, CVS	Git, Mercurial, Bazaar

4. Key Version Control Tools

A. Git

The undisputed industry standard, Git is a distributed system designed for speed and non-linear workflows.

- **Key Functionality:** Lightweight branching and merging. It uses a cryptographic hash (SHA-1) to ensure data integrity.
- **Role in SCM:** Enables **GitOps**. By integrating with CI/CD tools (like Jenkins or GitHub Actions), a "git push" can automatically trigger the configuration of servers and deployment of applications.
- **Best For:** Modern web development, DevOps teams, and open-source projects.

B. Apache Subversion (SVN)

A centralized system that was the dominant tool before Git. It acts like a "time machine" filesystem.

- **Key Functionality:** Directory-based branching (branches are just copies of folders). It supports "Path-based authorization," allowing granular access control (e.g., User A can read the whole repo but only write to /trunk).
- **Role in SCM:** useful for organizations requiring strict, centralized control over who can access specific sub-directories of a project.
- **Best For:** Enterprise legacy systems, projects requiring granular path-based permissions, or teams resistant to distributed workflows.

C. Perforce (Helix Core)

A centralized system optimized for massive scale and large binary files (images, audio, video).

- **Key Functionality:** "File Locking." Unlike Git, Perforce allows a user to "lock" a file so no one else can edit it—crucial for binary assets (like 3D models) that cannot be merged textually.
- **Role in SCM:** Manages configuration for large artifacts. It creates a seamless pipeline between artists (binaries) and coders (source text) in a single repository.
- **Best For:** Game development (Unreal Engine/Unity), movie production, and hardware simulation data.

D. Mercurial

A distributed system similar to Git but designed with a focus on simplicity and a simpler command-line interface.

- **Key Functionality:** Permanent, immutable history (harder to "rewrite" history than in Git), making it safer for beginners.
- **Role in SCM:** Provides a safer, less complex distributed configuration management experience, though its usage has declined significantly in favor of Git.
- **Best For:** Teams wanting distributed power without Git's steep learning curve (though now niche).

5. Functionality with Respect to Configuration Management

Version control tools provide the "How" for the "What" of Configuration Management.

1. Configuration Identification (Baselines)

VCS tools allow you to tag specific points in history (e.g., v1.0.0, Release-2025-Oct).

- **SCM Benefit:** You can instantly reproduce the exact state of the software *and* its configuration at any past moment. This is essential for debugging production issues.

2. Change Control & Auditing

Every change is recorded with metadata: **Who** made the change, **When**, and **Why** (commit message).

- **SCM Benefit:** Provides a complete audit trail for compliance. In a **GitOps** workflow, a "Pull Request" acts as a Change Control Board (CCB) meeting—changes are proposed, reviewed, and approved via code review before affecting the production configuration.

3. Infrastructure as Code (IaC) integration

Modern SCM does not just manage application code; it manages the environment (servers, networks, databases) via tools like **Terraform** or **Ansible**.

- **Functionality:** The VCS stores the text files that define the infrastructure.
- **SCM Benefit:** Prevents "Configuration Drift." If a server crashes, you don't rebuild it manually; you re-run the script stored in Version Control to restore the exact configuration.

Conclusion

For effective Configuration Management, the tool must match the asset type. **Git** is the superior choice for text-based assets (code, config scripts, documentation) and is the engine behind modern DevOps. However, **Perforce** remains widely relevant for industries managing heavy binary assets where merging is impossible.