What is Version Control

The IDEAS Scientific Software Productivity Project
ideas-productivity.org/resources/howtos/

Motivation: Software source, documentation, and other important (text) documents should be managed with a **Version Control System (VCS)** in order to do the following:

- Support safe incremental development (i.e., "undos")
- Support collaboration among different developers, contributors, and customers
- Provide traceability from requirements to file changes
- Streamline development and testing processes
- Provide reproducibility of past results

Users interact with a VCS through a formal process or **workflow**. In this document, we introduce some concepts and terminology of version control *[1]*, mention some of its benefits, describe use cases where it has been helpful (even critical), and outline some of the major VCS tools *[2]*.

Version Control Definitions and Terminology

Important terms and concepts in version control (see *[1]*) include the following.

- In a **Centralized VCS,** there is a single **repository**, and the user **workflow** consists of ***checking out*** stored versions, updating the checked-out copy, and ***checking in,*** *or* ***committing*** updated versions to the central repository. *Versions* of software are referred to as ***revisions*** or ***commits***. Users check out a single version at a time, and only the central repository stores all commits. If the central repository is lost without backups, the repository's entire history will be lost. Most VCS tools allow concurrent editing of the same files, requiring different versions to be merged. This may result in **merge conflicts**, which must be resolved manually before updating the central repository. (Note: merging and resolving merge conflicts can be risky and time consuming.)
- A **Distributed VCS** allows multiple complete copies of the same repository, and changes are moved back and forth between different repositories using various processes and workflows. Distributed VCS enables workflows that are well suited for build & test, code review, collaboration, and concurrent **branches**.
- A **branch** is an ordered set of commits representing a single history of changes to the files in a repository. Most systems support the creation and merging of branches. Branches are an especially important concept and tool in distributed VCS tools and processes.

Version Control Tools

Many high-quality open-source and commercial VCS are available *[2]*. Some of the more popular free open-source version control tools are the following:

- **Subversion (SVN)** is a popular centralized VCS started in 2000. The user interface is fairly simple and easy to learn. This is largely due to the simplicity of the centralized VC workflow, but branches are also supported.
- **Git** is a more recent distributed VCS started in 2005 to support the development of the Linux kernel. In recent years Git has become the most popular and dominant VCS in use, due to collaborative workflows enabled by distributed VCS and made popular

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.

by the *GitHub* code hosting site. Git has a large and complex user interface, and it takes significant effort to learn. Git is poorly suited for managing large binary data files (but extensions like *Git Large File Storage (LFS)* may help). Nevertheless, many large and complex projects use Git in all sectors of software development (but often use many smaller Git repositories rather than fewer large Git repositories).

- **Mercurial** (hg), another distributed VCS, started around 2005. It is less popular than Git but is generally considered to have a simpler user interface. Mercurial is used by many projects (e.g., it is the primary VCS tool for Facebook).
- **CVS** was the first popular, successful open-source VCS. It is an older centralized VCS, and Subversion was created as its successor. Although new projects no longer choose CVS, its influence and legacy are important to note when considering VCS tools and workflows.

Several tools exist to convert and interoperate between different open-source VCS (e.g., *git-svn*, *hg-git*). However, these tools vary greatly in maturity, performance, scalability, and usability.

Use Cases for Version Control

Version control is useful in numerous situations. Because most VCS tools are oriented to text file lines, users can manage and collaborate on nearly any set of text files in a VCS. The following are examples.

- **Software development by a single individual:** Enables the developer to keep track of older versions, support reproducibility of past results, pursue incremental commits with undo, simplify automated testing, help with porting, etc.
- **Software development by a team:** Aids in collaboration among developers, supports code reviews, provides traceability of requirements to code changes, etc.
- **Collaborative document writing:** For documents in plain text source (e.g., Latex, reStructuredText, Markdown, HTML), allows concurrent editing with merges (line by line), tracks who contributed to what sections (e.g., using "blame"), etc.

References:

[1] *https://en.wikipedia.org/wiki/Version_control*

[2] *https://en.wikipedia.org/wiki/List_of_version_control_software*

This document was prepared by Roscoe A. Bartlett with key contributions from Jim Willenbring and Todd Gamblin.

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.