A *version control system* (also known as a *Revision Control System*) is a repository of files, often the files for the source code of computer programs, with monitored access. Every change made to the source is tracked, along with who made the change, why they made it, and references to problems fixed, or enhancements introduced, by the change.

Version control systems are essential for any form of distributed, collaborative development. Whether it is the history of a wiki page or large software development project, the ability to track each change as it was made, and to reverse changes when necessary can make all the difference between a well-managed and controlled process and an uncontrolled ‘first come, first served’ system. It can also serve as a mechanism for due diligence for software projects.

**Version Tracking**

Developers may wish to compare today’s version of some software with yesterday’s version or last year’s version. Since version control systems keep track of every version of the software, this becomes a straightforward task. Knowing the what, who, and when of changes will help with comparing the performance of particular versions, working out when bugs were introduced (or fixed), and so on. Any problems that arose from a change can then be followed up by an examination of who made the change and the reasons they gave for making the change.

From : <http://oss-watch.ac.uk/resources/versioncontrol>  
------------------------------------------------------------------------------------------------------------------------------------------

## Version Control

A version control system (or revision control system) is a combination of technologies and practices for tracking and controlling changes to a project's files, in particular to source code, documentation, and web pages. If you have never used version control before, the first thing you should do is go find someone who has, and get them to join your project. These days, everyone will expect at least your project's source code to be under version control, and probably will not take the project seriously if it doesn't use version control with at least minimal competence.

The reason version control is so universal is that it helps with virtually every aspect of running a project: inter-developer communications, release management, bug management, code stability and experimental development efforts, and attribution and authorization of changes by particular developers. The version control system provides a central coordinating force among all of these areas. The core of version control is change management: identifying each discrete change made to the project's files, annotating each change with metadata like the change's date and author, and then replaying these facts to whoever asks, in whatever way they ask. It is a communications mechanism where a change is the basic unit of information.

This section does not discuss all aspects of using a version control system. It's so all-encompassing that it must be addressed topically throughout the book. Here, we will concentrate on choosing and setting up a version control system in a way that will foster cooperative development down the road.

### Version Control Vocabulary

This book cannot teach you how to use version control if you've never used it before, but it would be impossible to discuss the subject without a few key terms. These terms are useful independently of any particular version control system: they are the basic nouns and verbs of networked collaboration, and will be used generically throughout the rest of this book. Even if there were no version control systems in the world, the problem of change management would remain, and these words give us a language for talking about that problem concisely.

**"Version" Versus "Revision"**

The word version is sometimes used as a synonym for "revision", but I will not use it that way in this book, because it is too easily confused with "version" in the sense of a version of a piece of software—that is, the release or edition number, as in "Version 1.0". However, since the phrase "version control" is already standard, I will continue to use it as a synonym for "revision control" and "change control".

commit

To make a change to the project; more formally, to store a change in the version control database in such a way that it can be incorporated into future releases of the project. "Commit" can be used as a verb or a noun. As a noun, it is essentially synonymous with "change". For example: "I just committed a fix for the server crash bug people have been reporting on Mac OS X. Jay, could you please review the commit and check that I'm not misusing the allocator there?"

log message

A bit of commentary attached to each commit, describing the nature and purpose of the commit. Log messages are among the most important documents in any project: they are the bridge between the highly technical language of individual code changes and the more user-oriented language of features, bugfixes, and project progress. Later in this section, we'll look at ways to distribute log messages to the appropriate audiences; also, [the section called “Codifying Tradition”](http://producingoss.com/en/growth.html#codifying-tradition) in [Chapter 6, *Communications*](http://producingoss.com/en/communications.html) discusses ways to encourage contributors to write concise and useful log messages.

update

To ask that others' changes (commits) be incorporated into your local copy of the project; that is, to bring your copy "up-to-date". This is a very common operation; most developers update their code several times a day, so that they know they're running roughly the same thing the other developers are running, and so that if they see a bug, they can be pretty sure it hasn't been fixed already. For example: "Hey, I noticed the indexing code is always dropping the last byte. Is this a new bug?" "Yes, but it was fixed last week—try updating, it should go away."

repository

A database in which changes are stored. Some version control systems are centralized: there is a single, master repository, which stores all changes to the project. Others are decentralized: each developer has his own repository, and changes can be swapped back and forth between repositories arbitrarily. The version control system keeps track of dependencies between changes, and when it's time to make a release, a particular set of changes is approved for that release. The question of whether centralized or decentralized is better is one of the enduring holy wars of software development; try not to fall into the trap of arguing about it on your project lists.

checkout

The process of obtaining a copy of the project from a repository. A checkout usually produces a directory tree called a "working copy" (see below), from which changes may be committed back to the original repository. In some decentralized version control systems, each working copy is itself a repository, and changes can be pushed out to (or pulled into) any repository that's willing to accept them.

working copy

A developer's private directory tree containing the project's source code files, and possibly its web pages or other documents. A working copy also contains a little bit of metadata managed by the version control system, telling the working copy what repository it comes from, what "revisions" (see below) of the files are present, etc. Generally, each developer has his own working copy, in which he makes and tests changes, and from which he commits.

revision, change, changeset

A "revision" is usually one specific incarnation of a particular file or directory. For example, if the project starts out with revision 6 of file F, and then someone commits a change to F, this produces revision 7 of F. Some systems also use "revision", "change", or "changeset" to refer to a set of changes committed together as one conceptual unit.

These terms occasionally have distinct technical meanings in different version control systems, but the general idea is always the same: they give a way to speak precisely about exact points in time in the history of a file or a set of files (say, immediately before and after a bug is fixed). For example: "Oh yes, she fixed that in revision 10" or "She fixed that in revision 10 of foo.c."

When one talks about a file or collection of files without specifying a particular revision, it is generally assumed that one means the most recent revision(s) available.

diff

A textual representation of a change. A diff shows which lines were changed and how, plus a few lines of surrounding context on either side. A developer who is already familiar with some code can usually read a diff against that code and understand what the change did, and even spot bugs.

tag

A label for a particular collection of files at specified revisions. Tags are usually used to preserve interesting snapshots of the project. For example, a tag is usually made for each public release, so that one can obtain, directly from the version control system, the exact set of files/revisions comprising that release. Common tag names are things like Release\_1\_0, Delivery\_00456, etc.

branch

A copy of the project, under version control but isolated, so that changes made to the branch don't affect the rest of the project, and vice versa, except when changes are deliberately "merged" from one side to the other (see below). Branches are also known as "lines of development". Even when a project has no explicit branches, development is still considered to be happening on the "main branch", also known as the "main line" or "trunk".

Branches offer a way to isolate different lines of development from each other. For example, a branch can be used for experimental development that would be too destabilizing for the main trunk. Or conversely, a branch can be used as a place to stabilize a new release. During the release process, regular development would continue uninterrupted in the main branch of the repository; meanwhile, on the release branch, no changes are allowed except those approved by the release managers. This way, making a release needn't interfere with ongoing development work. See [the section called “Use branches to avoid bottlenecks”](http://producingoss.com/en/vc.html#branches) later in this chapter for a more detailed discussion of branching.

merge (a.k.a. port)

To move a change from one branch to another. This includes merging from the main trunk to some other branch, or vice versa. In fact, those are the most common kinds of merges; it is rare to port a change between two non-main branches. See [the section called “Singularity of information”](http://producingoss.com/en/vc.html#vc-singularity) for more about this kind of merging.

"Merge" has a second, related meaning: it is what the version control system does when it sees that two people have changed the same file but in non-overlapping ways. Since the two changes do not interfere with each other, when one of the people updates their copy of the file (already containing their own changes), the other person's changes will be automatically merged in. This is very common, especially on projects where multiple people are hacking on the same code. When two different changes do overlap, the result is a "conflict"; see below.

conflict

What happens when two people try to make different changes to the same place in the code. All version control systems automatically detect conflicts, and notify at least one of the humans involved that their changes conflict with someone else's. It is then up to that human to resolve the conflict, and to communicate that resolution to the version control system.

lock

A way to declare an exclusive intent to change a particular file or directory. For example, "I can't commit any changes to the web pages right now. It seems Alfred has them all locked while he fixes their background images." Not all version control systems even offer the ability to lock, and of those that do, not all require the locking feature to be used. This is because parallel, simultaneous development is the norm, and locking people out of files is (usually) contrary to this ideal.

Version control systems that require locking to make commits are said to use the lock-modify-unlock model. Those that do not are said to use the copy-modify-merge model. An excellent in-depth explanation and comparison of the two models may be found at [svnbook.red-bean.com/svnbook-1.0/ch02s02.html](http://svnbook.red-bean.com/svnbook-1.0/ch02s02.html). In general, the copy-modify-merge model is better for open source development, and all the version control systems discussed in this book support that model.

### Choosing a Version Control System

As of this writing, the two most popular version control systems in the free software world are Concurrent Versions System (CVS, [cvs.nongnu.org](http://cvs.nongnu.org/)) and Subversion (SVN, [subversion.apache.org](http://subversion.apache.org/)).

CVS has been around for a long time. Most experienced developers are already familiar with it, it does more or less what you need, and since it's been popular for a long time, you probably won't end up in any long debates about whether or not it was the right choice. CVS has some disadvantages, however. It doesn't provide an easy way to refer to multi-file changes; it doesn't allow you to rename or copy files under version control (so if you need to reorganize your code tree after starting the project, it can be a real pain); it has poor merging support; it doesn't handle large files or binary files very well; and some operations are slow when large numbers of files are involved.

None of CVS's flaws is fatal, and it is still quite popular. However, in the last few years the more recent Subversion has been gaining ground, especially in newer projects.[[20](http://producingoss.com/en/vc.html" \l "ftn.idp5901680)]. If you're starting a new project, I recommend Subversion.

On the other hand, since I'm involved in the Subversion project, my objectivity might reasonably be questioned. And in the last few years a number of new open-source version control systems have appeared. [Appendix B, *Free Version Control Systems*](http://producingoss.com/en/vc-systems.html) lists all the ones I know of, in rough order of popularity. As the list makes clear, deciding on a version control system could easily become a lifelong research project. Possibly you will be spared the decision because it will be made for you by your hosting site. But if you must choose, consult with your other developers, ask around to see what people have experience with, then pick one and run with it. Any stable, production-ready version control system will do; you don't have to worry too much about making a drastically wrong decision. If you simply can't make up your mind, then go with Subversion. It's fairly easy to learn, and is likely to remain a standard for at least a few years.

From: http://producingoss.com/en/vc.html

------------------------------------------------------------------------------------------------------------------------------------------

## Version Control Basics

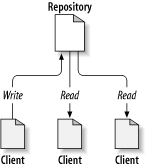
A version control system (or revision control system) is a system that tracks incremental versions (or revisions) of files and, in some cases, directories over time. Of course, merely tracking the various versions of a user's (or group of users') files and directories isn't very interesting in itself. What makes a version control system useful is the fact that it allows you to explore the changes which resulted in each of those versions and facilitates the arbitrary recall of the same.

In this section, we'll introduce some fairly high-level version control system components and concepts. We'll limit our discussion to modern version control systems—in today's interconnected world, there is very little point in acknowledging version control systems which cannot operate across wide-area networks.

### The Repository

At the core of the version control system is a repository, which is the central store of that system's data. The repository usually stores information in the form of a filesystem tree—a hierarchy of files and directories. Any number of clients connect to the repository, and then read or write to these files. By writing data, a client makes the information available to others; by reading data, the client receives information from others. [Figure 1.1, “A typical client/server system”](http://svnbook.red-bean.com/en/1.7/svn.basic.version-control-basics.html#svn.basic.repository.dia-1) illustrates this.

**Figure 1.1. A typical client/server system**



Why is this interesting? So far, this sounds like the definition of a typical file server. And indeed, the repository is a kind of file server, but it's not your usual breed. What makes the repository special is that as the files in the repository are changed, the repository remembers each version of those files.

When a client reads data from the repository, it normally sees only the latest version of the filesystem tree. But what makes a version control client interesting is that it also has the ability to request previous states of the filesystem from the repository. A version control client can ask historical questions such as “What did this directory contain last Wednesday?” and “Who was the last person to change this file, and what changes did he make?” These are the sorts of questions that are at the heart of any version control system.

### The Working Copy

A version control system's value comes from the fact that it tracks versions of files and directories, but the rest of the software universe doesn't operate on “versions of files and directories”. Most software programs understand how to operate only on a single version of a specific type of file. So how does a version control user interact with an abstract—and, often, remote—repository full of multiple versions of various files in a concrete fashion? How does his or her word processing software, presentation software, source code editor, web design software, or some other program—all of which trade in the currency of simple data files—get access to such files? The answer is found in the version control construct known as a working copy.

A working copy is, quite literally, a local copy of a particular version of a user's VCS-managed data upon which that user is free to work. Working copies[[5](http://svnbook.red-bean.com/en/1.7/svn.basic.version-control-basics.html#ftn.idp5943408)] appear to other software just as any other local directory full of files, so those programs don't have to be “version-control-aware” in order to read from and write to that data. The task of managing the working copy and communicating changes made to its contents to and from the repository falls squarely to the version control system's client software.

From: http://svnbook.red-bean.com/en/1.7/svn.basic.version-control-basics.html

------------------------------------------------------------------------------------------------------------------------------------------

To learn more: <http://git-scm.com/video/what-is-version-control>

------------------------------------------------------------------------------------------------------------------------------------------