**Concurrent Versions System (CVS)**

[CVS](http://www.nongnu.org/cvs/) has been around since the 80s, and has been very popular with both commercial and open source developers.

It is released under the GNU license, and uses a system to let users “check out” the code they are going to work on and “check in” their changes.

Originally, CVS handled conflicts between two programmers by only allowing for the latest version of the code to be worked on and updated. As such, it was a first come, first serve system where the user must publish changes quickly to ensure that other users haven’t beat them to the punch.

Now, CVS can handle branching projects so the developed software can diverge into different products with unique features and will be reconciled at a later time.

The CVS server runs on Unix-like systems with client software that runs on multiple operating systems. It is considered the most mature version control system because it has been developed for such a long time and does not receive many requests for new features at this time.

A fork project of CVS, CVSNT was created to run CVS on Windows servers, and it is currently being actively developed to increase functionality.

**Pros:**

* Has been in use for many years and is considered mature technology

**Cons:**

* Moving or renaming files does not include a version update
* Security risks from symbolic links to files
* No atomic operation support, leading to source corruption
* Branch operations are expensive as it is not designed for long-term branching

**Apache Subversion (SVN)**

Apache Subversion

[SVN](http://subversion.apache.org/) was created as an alternative to CVS that would fix some bugs in the CVS system while maintaining high compatibility with it.

Like CVS, SVN is free and open source with the difference of being distributed under the Apache license as opposed to GNU.

To prevent corruption in the database from being corrupted, SVN employs a concept called atomic operations. Either all of the changes made to the source are applied or none are applied, meaning that no partial changes will break the original source.

Many developers have switched to SVN as it is a newer technology that takes the best features of CVS and improves upon them.

While CVS’s branch operations are expensive and do not really lend themselves to long-term forks in the project, SVN is designed to allow for it, lending itself better to large, forked projects with many directions.

Criticism of SVN includes slower comparative speed and the lack of distributed revision control. Distributed revision control uses a peer-to-peer model rather than using a centralized server to store code updates. While a peer-to-peer model would work better for world-wide, open source projects, it may not be ideal in other situations. The downside to a dedicated server approach is that when the server is down, no clients are able to access the code.

**Pros:**

* Newer system based on CVS
* Includes atomic operations
* Cheaper branch operations
* Wide variety of plug-ins for IDEs
* Does not use peer-to-peer model

**Cons:**

* Still contains bugs relating to renaming files and directories
* Insufficient repository management commands
* Slower comparative speed

**Git**



First developed by Linus Torvalds of Linux fame, [Git](http://git-scm.com/) takes a radical approach that differs greatly from CVS and SVN.

The original concepts for Git were to make a faster, distributed revision control system that would openly defy conventions and practices used in CVS. It is primarily developed for Linux and has the highest speeds on there.

It will also run on other Unix-like systems, and native ports of Git are available for Windows as msysgit.

As there is no centralized server, Git does not lend itself to single developer projects or small teams as the code may not necessarily be available when using a non-repository computer. Workarounds exist for this problem, and some see Git’s improved speed as a decent tradeoff for the hassle.

Git also comes equipped with a wide variety of tools to help users navigate the history system. Each instance of the source contains the entire history tree, which can be useful when developing without an internet connection.

**Pros:**

* Great for those who hate CVS/SVN
* Dramatic increase in operation speed
* Cheap branch operations
* Full history tree available offline
* Distributed, peer-to-peer model

**Cons:**

* Learning curve for those used to SVN
* Not optimal for single developers
* Limited Windows support compared to Linux

**Mercurial**



[Mercurial](http://mercurial.selenic.com/) began close to the same time as Git and is also a distributed revision control tool.

It was originally made to compete with Git for Linux kernel development, and as Git was selected, Mercurial has seen less success in that area. However, that is not to say that it is not used as many major developments use it, including [OpenOffice.org](http://www.openoffice.org/).

It’s different from other revision control systems in that Mercurial is primarily implemented in Python as opposed to C, but there are some instances where C is used.

Due to its distributed nature and its creation in Python, the Python language developers are considering a switch to Mercurial as it would allow non-core developers to have easier access to creating new trees and reverting changes.

Users have noted that Mercurial shares some features with SVN as well as being a distributed system, and because of the similarities, the learning curve for those already familiar with SVN will be less steep. The documentation for Mercurial also is more complete and will facilitate learning the differences faster.

Some of the major drawbacks to Mercurial include that it doesn’t allow for two parents to be merged and unlike Git, it uses an extension system rather than being scriptable. That may be ideal for some programmers, but many find the power of Git to be a feature they don’t want to trade off.

**Pros:**

* Easier to learn than Git
* Better documentation
* Distributed model

**Cons:**

* No merging of two parents
* Extension-based rather than scriptability
* Less out of the box power

**Which system should I use?**



For the most part, people use CVS because they are already used to it, and while some of the quirks and limitations are bothersome, they’ve already figured out how to make a work around the given limitations and have those workarounds implemented. Especially if your team is currently engaged on a certain project, the prospect of migrating everything to another revision control is annoying, and if they were to switch, it would most likely be to SVN.

As it has now moved into the “mature technology” part of its life cycle, it is unlikely that CVS will come out with any groundbreaking features, and as momentum is lost in the project as people move to SVN, it appears most likely it is on its way out.

SVN is currently the king of server-based version control. It has all of the good features of CVS and improves upon them. In terms of corporate interaction, you are more likely to come across CVS or SVN than you will with Git or Mercurial, so a familiarity with single server technology, while not a requirement, will ease the transitions in the workplace.

With its wide range of usage and its software maturity level, SVN has a large knowledge base, and users will be able to find help readily accessible from other users.

Git has a clear speed improvement over its competitors, and for projects that lend themselves to distributed systems, it is a clear improvement.

The primary downside cited for Git is that it can be at times difficult to explain to others, and there is likely to be a slow down in production as programmers adapt to it. Once it is learned, however, the speed increases and better branch management will reclaim that time and more.

For those absolutely repulsed by Git (and it does have its sworn enemies in the programming world), Mercurial offers a bridge between SVN and Git that is well documented and used in many well known projects.

The Windows-friendly version of Git has also made some strides which brings the speed closer to that of the Linux versions, so it could still be on the table if you are not developing in Linux.

To find out which one is best for you, consider the project and the developers. And talk to them!

If you want to have a single master source tree that is being worked on by a small core development group, SVN should be the first system you try as it’s reliable and tailored for that.

If you are starting an open source project where several programmers are going to be working at different times and/or submitting several updates to the code, Git is an excellent choice for your project due to the huge speed boost and improved tree management over SVN.

If you’re somewhere in the middle or you just really don’t like the way SVN or Git works for you after trying them, there’s always Mercurial.

All of these systems are fully functional. They’re also all free. They’ve all been used to create software, websites, and even operating systems that you’ve used and heard of.

The first decision you will need to make is whether the choice fit the needs of your business and team, and then—just as important—you should make sure your choice isn’t going to send your coding team into a rage.

**Getting Started with SVN**

If you’ve never worked with SVN or GIT before, and wouldn’t have a clue where to start, a hosted solution coupled with a graphical desktop client means you can be up and running in no time.

Like most things, you’ll learn your way around more quickly by jumping in and trying things out – Using Subversion is very similar to working with files from your host using an FTP client. If you’ve never done that before, perhaps try that first.

**NOTE:** I wouldn’t have a clue how to set up subversion hosting, there are plenty of great hosted options around, some with free trials so you can create your very first repository (the place where you’ll collaborate on files) at no cost. Here are some options to check out.

**SVN & GIT Hosting**

* [[](http://beanstalkapp.com/)](http://beanstalkapp.com/)

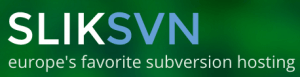
Beanstalk

– Secure, private and reliable[Git](http://git-scm.com/) and Subversion hosting. View activity, browse files, compare revisions. Great user interface. Integrates with a bunch of popular services, including Twitter! **From $15 a month**

* [[](http://unfuddle.com/)](http://unfuddle.com/)

Unfuddle

– popular subversion and git hosting and hosted project management solution for software development teams. Comes with a handy Widget for Mac OS – you can monitor account activity across all of your projects from one simple interface. **Free with 2 people, with 1 project for up to 200MB storage**

* [[](http://www.sliksvn.com/)](http://www.sliksvn.com/)

SlikSVN

– another option for free hosting, but not as good as Unfuddle, subscriptions are competitive. **Free with 1 person, with 1 project for up to 100MB storage**

* [[XP-Dev.com](http://xp-dev.com/)](http://xp-dev.com/)

XP-Dev.com

– Subversion, GIT and Mercurial hosting. A decent feature set and represent excellent value for money. **From $5 a month with unlimited projects, users and up to 2GB of storage**

**Creating your first repository**

Once you’ve created an account, you will need to create a repository – different on each platform. Typically you’d do something like this:

* Once logged in, click on your projects
* Create a project:
  + Under Create a New Project enter a name for your project.
  + Click on the Create Project button
* Enable Subversion:
  + Once your project is created, click on the Source Control tab
  + Click on the link Enable Source Control
  + Give your Subversion repository a name.
  + Click on the Save button

**Graphical Clients for SVN and GIT**

You’ve got your repository setup, and if you want a simple way to access it, a graphical interface is your best option.

* TortoiseSVN subversion client

TortoiseSVN

– an easy to use source control software for Microsoft Windows and possibly the best standalone Apache Subversion client there is. It is implemented as a Windows shell extension, which makes it integrate seamlessly with explorer. **It’s also open source with around 34 language packs available**

* [[](https://www.syntevo.com/smartgit/)](https://www.syntevo.com/smartgit/)

SmartGit

– a graphical client for Git, an Open Source distributed version control system. It runs on Windows, Mac OS X and Linux. **$59/year.**

* [[](http://versionsapp.com/)](http://versionsapp.com/)

Versions

– SVN client for MAC OS X – Versions offers the simplest Subversion on the Mac. In their own words, “Thanks to its clear-cut approach, you’ll hit the ground running”. It*is* easy to use. **$39 GBP (around $64 USD at today’s exchange rates).**

* [[](https://www.git-tower.com/)](https://www.git-tower.com/)

Tower

– GIT client for MAC OS X – is another very elegant solution, and is perfect for GIT users who have no need to learn command line syntax.  There’s also a good introductory overview on their product tour which covers a lot of the basic GIT concepts (see below). **$69/user/year.**

**“Checkout” your repository**

In whatever client you are using, you need to create a new source control repository. You will need to enter the URL of you repository along with your user name and password in your client.

The URL will typically look something like: *https://svn<server number>.hostname.com/svn/<repository>*  
(you can use https:// (SSL) if you have a paid account)

1. click on the root folder, then click on the Check Out but­ton and cre­ate a work­ing folder in the client. You can now add files to this folder.
2. After check­ing out the project files, you can edit them from the directory you’ve created on your local computer

After edit­ing files, to commit them, click the check In but­ton in the tool­bar to update the changes. You can review the changes and add a com­ment – it’s a good idea to add comments so you can remember what you were work­ing on, what changes you made, and to notify other members of your project what you’ve been up to.

In your client you should also be able to check revisions at any time by click­ing on the revision viewer or history button – you can then restore previous changes, for each file if necessary. Once you have your head around these basic concepts, the documentation provided with each client is a good starting point for more advanced features.

All of these systems are fully functional. They’re also all free. They’ve all been used to create software, websites, plugins and [themes](https://www.collectiveray.com/wordpress/wordpress-themes/divi-theme-review.html), coding frameworks, mobile and web applications and even fully-fledged operating systems that you’ve used and heard of.

Besides the ability to back up your code safely in the cloud, allowing your developers to work from anywhere, version control can also allow you to find and revert changes which might have broken your regression tests. By performing a diff between the current and previous versions, you can easily determine what has changes were made and then rectify as necessary.