Control Version System (SVN, CSV, Git, Mercurial etc)

Система контроля версий упрощает совместную разработку, поскольку позволяет удобно обмениваться наработками, снижает вероятность потери данных, существования конфликтов

СКВ – позволяет большому количеству разработчиков работать с одним и тем же кодом, переходить между разными коммитами, смотреть историю файла, кем был внесены изменения, отменить изменения

**Following are the goals of a Version Control System.**

* Allow developers to work simultaneously.
* Do not overwrite each other’s changes.
* Maintain history of every version of everything.

**A VCS is divided into two categories.**

* Centralized Version Control System (CVCS), meaning that it uses central server to store all files and enables team collaboration.
* Distributed/Decentralized Version Control System (DVCS).

CVS (Concurrent Versioning System )– самая первая СКВ и считается «стандартом»

CVS – централизированная СКВ

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Compared item** | **CVS** | **SVN** |
| 2 | Speed | CVS works more slowly. | As a whole, due to some constructive solutions, SVN really works faster than CVS. It transmits less information through the network and supports more operations for offline mode. However, there is the reverse of the medal. Speed increasing is achieved basically at the expense of full backup of all work files on your computer. |
| 5 | File types | CVS was initially intended for text data storage. That is why storage of other files (binary, unicode) is not trivial and requires special information, as well as adjustments on either server or client sides. | SVN manipulates all the file types and does not require your instructions. |
| 6 | Rollback | CVS allows to rollback any commit in the repository, even if this may require some time (each file should be processed independently) | SVN does not allow rollback of commit. Authors suggest copy good repository state to the end of trunk to overwrite bad commit. However bad commit itself will remain in repository. |
| 7 | Transactions | In CVS the support of transactions by the principle "all or nothing" is completely absent. For example, when you checkin several files (transfer them to the server), it is possible that the operation will be completed only for some of these files, and will not be completed for the rest (due to conflicts, for example). As a rule, it is sufficient to correct the situation and to repeat the operation for the remaining files (not for all files). That is, the files will be checked in in two steps. No cases of the repository damage due to absence of this functionality were observed. | SVN does support transactions by the principle "all or nothing". Probably this is an advantage of the system. |
| 8 | Availability | Presently CVS is supported everywhere where you might need it. | SVN not yet so widly used, as the result there are places where it support still not implemented. |
| 9 | Internal architecture and code. | CVS is very old system. Initially CVS was written as bunch of scripts around RCS executable. Later this packaged into single executable. But internal structure of code is poor and have lots of historical fixes. Till now there was several attempts to rewrite CVS from scratch, but as it is known without success. We, personally, tried to extract client code to make better integration between plug-in and CVS, but without success. Right now we not think that CVS may grows too much in its functionality. | Subversion authors really spent some time on internal SVN architecture. Still not know how good are some decisions they make. But one is clear, the code is well expandable, and future improvements are coming. |

 Git offers a much different type of version control in that it’s a **distributed version control system**. With a distributed version control system, there isn’t one centralized code base to pull the code from. Different branches hold different parts of the code. Other version control systems, such as SVN and CVS, use centralized version control, meaning that only one master copy of the software is used.

[**Слепки вместо патчей**](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git#Слепки-вместо-патчей)

Главное отличие Git'а от любых других СКВ (например, Subversion и ей подобных) — это то, как Git смотрит на свои данные. В принципе, большинство других систем хранит информацию как список изменений (патчей) для файлов. Эти системы (CVS, Subversion, Perforce, Bazaar и другие) относятся к хранимым данным как к набору файлов и изменений, сделанных для каждого из этих файлов во времени, как показано на рисунке 1-4.



Рисунок 1-4. Другие системы хранят данные как изменения к базовой версии для каждого файла.

Git не хранит свои данные в таком виде. Вместо этого Git считает хранимые данные набором слепков небольшой файловой системы. Каждый раз, когда вы фиксируете текущую версию проекта, Git, по сути, сохраняет слепок того, как выглядят все файлы проекта на текущий момент. Ради эффективности, если файл не менялся, Git не сохраняет файл снова, а делает ссылку на ранее сохранённый файл. То, как Git подходит к хранению данных, похоже на рисунок 1-5.



Рисунок 1-5. Git хранит данные как слепки состояний проекта во времени.

Это важное отличие Git'а от практически всех других систем контроля версий. Из-за него Git вынужден пересмотреть практически все аспекты контроля версий, которые другие системы переняли от своих предшественниц. Git больше похож на небольшую файловую систему с невероятно мощными инструментами, работающими поверх неё, чем на просто СКВ. В главе 3, коснувшись работы с ветвями в Git'е, мы узнаем, какие преимущества даёт такое понимание данных.

[**Git следит за целостностью данных**](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git#Git-следит-за-целостностью-данных)

Перед сохранением любого файла Git вычисляет контрольную сумму, и она становится индексом этого файла. Поэтому невозможно изменить содержимое файла или каталога так, чтобы Git не узнал об этом. Эта функциональность встроена в сам фундамент Git'а и является важной составляющей его философии. Если информация потеряется при передаче или повредится на диске, Git всегда это выявит.

Механизм, используемый Git'ом для вычисления контрольных сумм, называется SHA-1 хешем. Это строка из 40 шестнадцатеричных символов (0-9 и a-f), вычисляемая в Git'е на основе содержимого файла или структуры каталога. SHA-1 хеш выглядит примерно так:

24b9da6552252987aa493b52f8696cd6d3b00373

Работая с Git'ом, вы будете встречать эти хеши повсюду, поскольку он их очень широко использует. Фактически, в своей базе данных Git сохраняет всё не по именам файлов, а по хешам их содержимого.

### [Чаще всего данные в Git только добавляются](https://git-scm.com/book/ru/v1/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git#Чаще-всего-данные-в-Git-только-добавляются)

Практически все действия, которые вы совершаете в Git'е, только добавляют данные в базу. Очень сложно заставить систему удалить данные или сделать что-то неотменяемое. Можно, как и в любой другой СКВ, потерять данные, которые вы ещё не сохранили, но как только они зафиксированы, их очень сложно потерять, особенно если вы регулярно отправляете изменения в другой репозиторий.