# Лекция 2. Subversion. Основные сведения.

## 2.1. Общие сведения.

Subversion — это свободная система управления версиями с открытым исходным кодом. Согласно классификации из предыдущей лекции, является централизованной.

Subversion позволяет управлять файлами и каталогами во времени. Дерево файлов помещается в центральное хранилище, которое похоже на обычный сервер файлов с тем отличием, что оно запоминает каждое изменение, внесённое в файл или каталог. Это позволяет восстановить ранние версии данных, исследовать историю изменений данных. Благодаря этому, многие считают систему управления версиями своеобразной «машиной времени».

## 2.2. История создания.

В начале 2000 года компания CollabNet, Inc решила начать разработку ПО, призванного прийти на смену CVS сохранив её основные идеи и устанив недостатки. Также разработчики хотели упростить переход от CVS к новой системе, сделав их схожими.

В феврале 2000 года CollabNet связалась с автором книги «Open Source Development with CVS» Карлом Фогелем, которые впоследствие не только придумал название «Subversion», но и разработал основные принципы устройства хранилища.

Карл Фогелеь и Бена Коллинз-Сассман в команде CollabNet в мае начали работать над проектированию системы. Вокруг Subversion быстро образовалось сообщество активных разработчиков, т.к. многие были откровенно недовольны CVS.

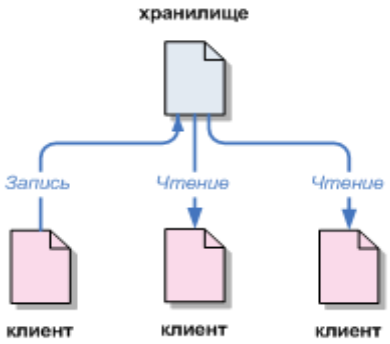
31 августа 2001 года, спустя четырнадцать месяцев с начала работы, команда прекратила использовать CVS и перешла на Subversion для управления версиями собственного исходного кода.

Сейчас SVN развивается как Open Sourse проект, но CollabNet так же оплачивает нескольким ключевым сотрудникам поддержку SVN.

## 2.3. Основные понятия.

Subversion является централизованной системой для разделения информации. В ее основе хранилище, являющееся центром хранения данных. Хранилище хранит информацию в форме *дерева файлов* — типичном представлении файлов и каталогов. Любое количество клиентов подключается к хранилищу и читает или записывает эти файлы. Записывая данные, клиент

делает информацию доступной для остальных; читая данные клиент получает информацию от других.




*Типичная клиент - серверная система*

Выглядит как определение типичного файл - сервера. Хранилище Subversion также запоминает каждое внесенное изменение: любое изменение любого файла, равно как изменения в самом дереве каталогов, такие как добавление, удаление и реорганизация файлов и каталогов.

При чтении данных из хранилища клиент обычно видит только последнюю версию дерева файлов. Но клиент также имеет возможность просмотреть предыдущие состояния файловой системы. Например, клиент может запросить такие данные как, «Что содержал этот каталог в прошлую среду?» или «Кто был последним изменявшим этот файл и какие вносились изменения?».

*Рабочая копия* Subversion представляет собой обычное дерево каталогов на вашем компьютере, содержащее набор файлов. Вы можете по-своему усмотрению редактировать эти файлы и, если это исходные коды, вы можете обычным способом скомпилировать программу. Ваша рабочая копия это ваше личное рабочее пространство: Subversion как не смешивает вносимые другими изменения с вашими, так и не делает доступными для других изменения сделанные вами, пока вы не прикажете сделать это. Вы даже можете иметь несколько рабочих копий одного и того же проекта.

Рабочая копия содержит несколько дополнительных файлов, созданных и обслуживаемых Subversion, которые помогают ей при выполнении команд. В частности, каждый каталог в вашей рабочей копии содержит подкаталог с именем .svn который называется *служебным каталогом рабочей копии*. Файлы в служебном каталоге помогают Subversion определить какие файлы рабочей копии содержат неопубликованные изменения, и какие файлы устарели по отношению к файлам других участников.

**Дельта — кодирование**

Subversion использует дельта — кодирование для компактного хранения изменений между версиями хранилища.

*Дельта-кодирование (Delta encoding)* — способ представления данных в виде разницы (дельты) между последовательными данными вместо самих данных.

**Как рабочие копии отслеживают хранилище**

В служебном каталоге .svn/ для каждого файла рабочего каталога Subversion записывает информацию о двух важнейших свойствах: на какой правке основан ваш рабочий файл (это называется *рабочая правка* файла), и *временной метке*, определяющей, когда рабочая копия последний раз обновлялась из хранилища.

Используя эту информацию при соединении с хранилищем, Subversion может сказать, в каком из следующих четырех состояний находится рабочий файл:

- Не изменялся и не устарел
- Изменялся локально и не устарел
- Не изменялся и устарел
- Изменялся локально и устарел (файл необходимо сначала обновить)

## 2.3. Модели версионирования

Разные СКВ используют разные способы для достижения успешного совместного редактирования. Проблема разделения файлов состоит в том, чтобы предоставить пользователям возможность совместного использования информации, при этомне наступая друг другу на пятки. Пользователи могут просто непреднамеренно перезаписать в хранилище изменения друг друга.

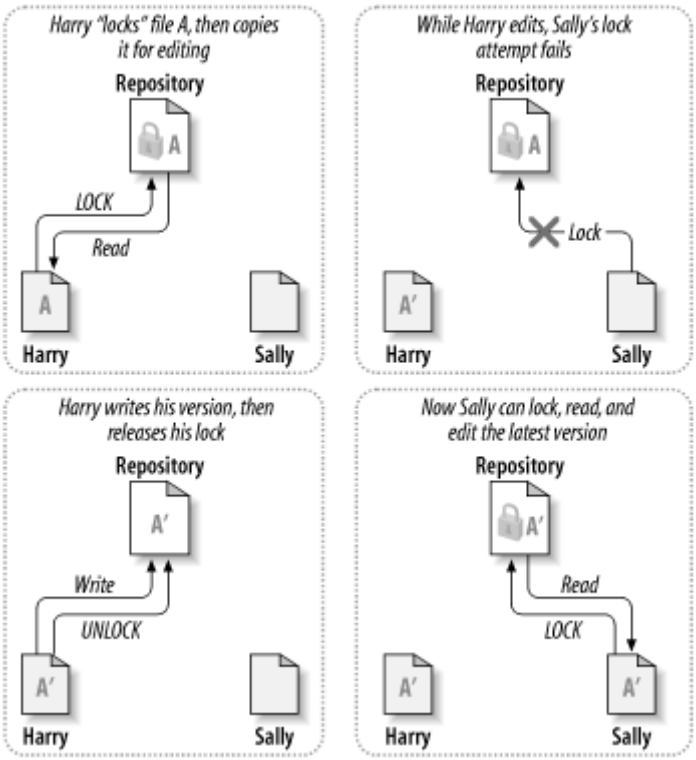
Чтобы этого не происходило, придумано множество моделей работы с файлами. Рассмотрим некоторые из них.

Существующие модели:

- Блокирование - Изменение - Разблокирование;
- Копирование — Изменение - Слияние.

### 2.3.1. Модель «Блокирование — изменение — разблокирование».

Описание модели: пользователь, решивший редактировать файл, блокирует его, в то время другой пользователь не может его редактировать. При разблокировке файла следующим участником другой пользователь может забрать изменения предыдущего пользователя и редактировать файл дальше.



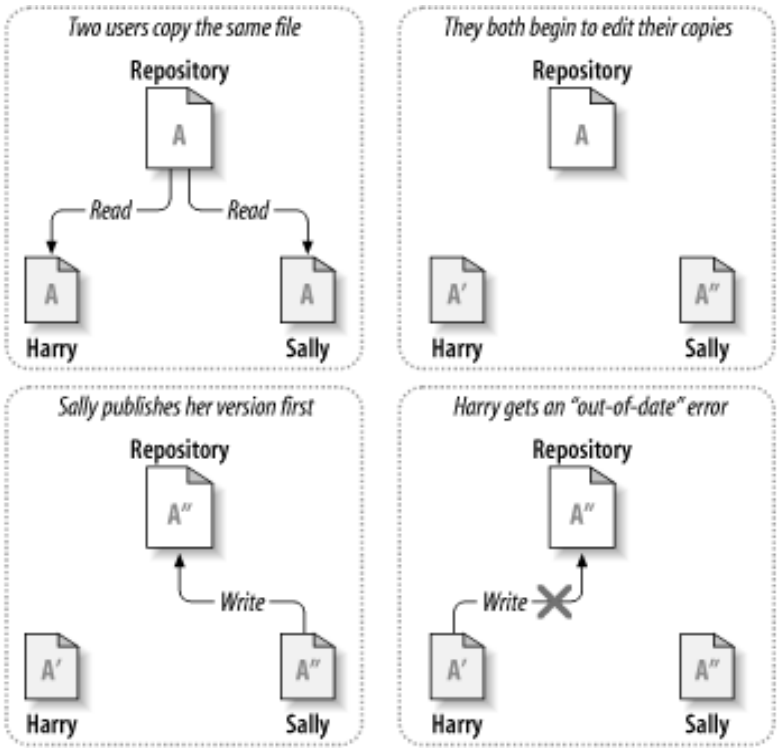

*Пример модели "блокирование-изменение-разблокирование"*

Проблемы модели:

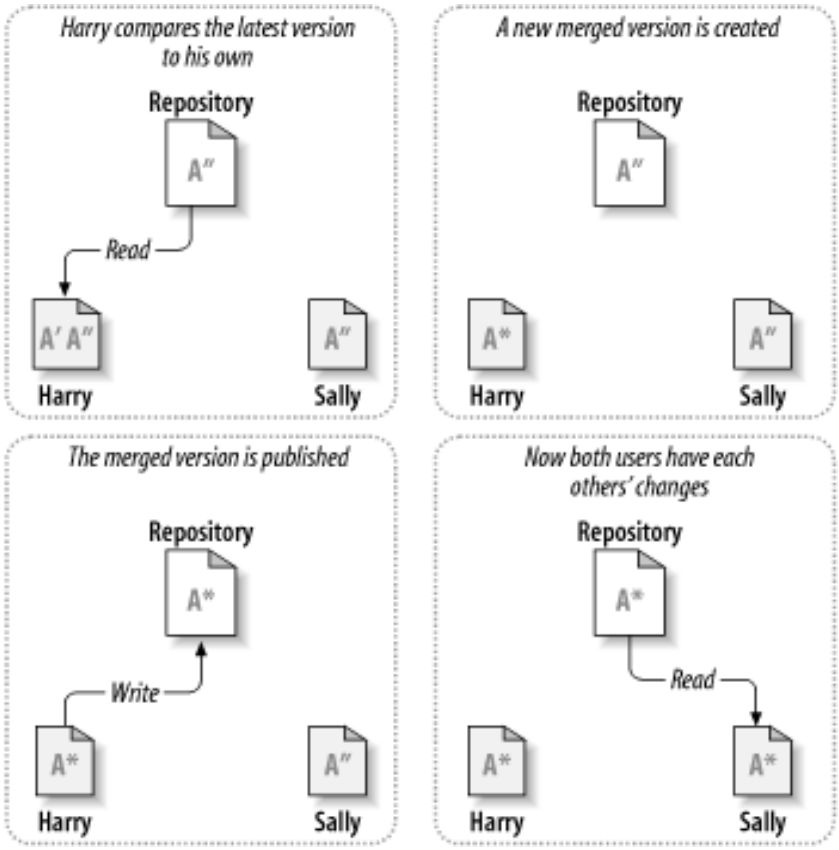
- Блокирование может вызвать проблемы администрирования. Файл может быть заблокирован, и забыт быть разблокированным. Придётся обращаться к администратору. Ведёт к потерянному времени;
- Блокирование может вызвать излишнюю пошаговость, например, если участники проекта хотят редактировать разные, не перекрывающиеся части файла;
- Блокирование может вызвать ложное чувство безопасности. Если файлы A и B редактируются двумя людьми, но эти файлы зависят друг от друга, изменения всё равно приведут к поломкам.

### 2.3.2. Модель «копирование — изменение — слияние».

Описание модели: оба пользователя копируют файл и редактируют его на своём компьютере. Затем один из пользователей отправляет изменения на сервер. Второй пользователь заканчивает свои изменения и тоже хочет отправить файлы на сервер, но не может, т.к. там уже находится обновлённая версия. Он должен сначала взять изменения с сервера, объединить изменения, разрешить конфликты, если они есть, и затем отправить свои изменения на сервер.




*Пример модели "копирование-изменение-слияние", часть 1*




*Пример модели "копирование-изменение-слияние", часть 2*

Subversion, CVS и другие системы управления версиями пользуются моделью копирование-изменение-слияние в качестве альтернативы блокированию. В этой модели каждый пользовательский клиент связывается с хранилищем проекта и создаёт персональную рабочую копию — локальное отражение файлов и каталогов хранилища. После этого пользователи работают параллельно, изменяя свои личные копии. В конце концов, личные копии сливаются в новую, финальную версию. Обычно система управления версиями помогает в слиянии, но, разумеется, за его корректное выполнение отвечает человек.

Плюсы подхода:

- Пользователи могут работать параллельно, не тратя время на ожидание друг друга;
- При работе над одними и теми же файлами оказывается, что большинство параллельно вносимых изменений совсем не перекрываются, конфликты бывают редко;
- время, которое было потрачено на разрешение конфликтов значительно меньше времени отнимаемого блокирующей системой.

Минусы подхода:

- немного сложнее для понимания, чем первая модель. Но, несмотря на это, отлично работает;

Не стоит возлагать большие надежды на то, что блокирующая система лучше защищена от конфликтов; на практике блокирование снижает продуктивность как ничто другое.

Несмотря на то, что модель блокирование — изменение - разблокирование в целом, губительна для командной работы, всё-таки есть моменты когда блокирование уместно, например для файлов бинарных форматов, таких как графические или звуковые, как правило не возможно объединить конфликтующие изменения. В таких ситуациях пользователям действительно необходимо быть внимательными при изменении файла. Без раздельного доступа кто-то может впустую потратить время на изменения, которые в конце концов будут потеряны.

### 2.3 Полезные команды

**svn help**

Клиент для командной строки Subversion является самодокуметируемым — в любой момент команда *svn help <subcommand>* покажет описание синтаксиса, параметров и поведения подкоманды subcommand.

**svn info**

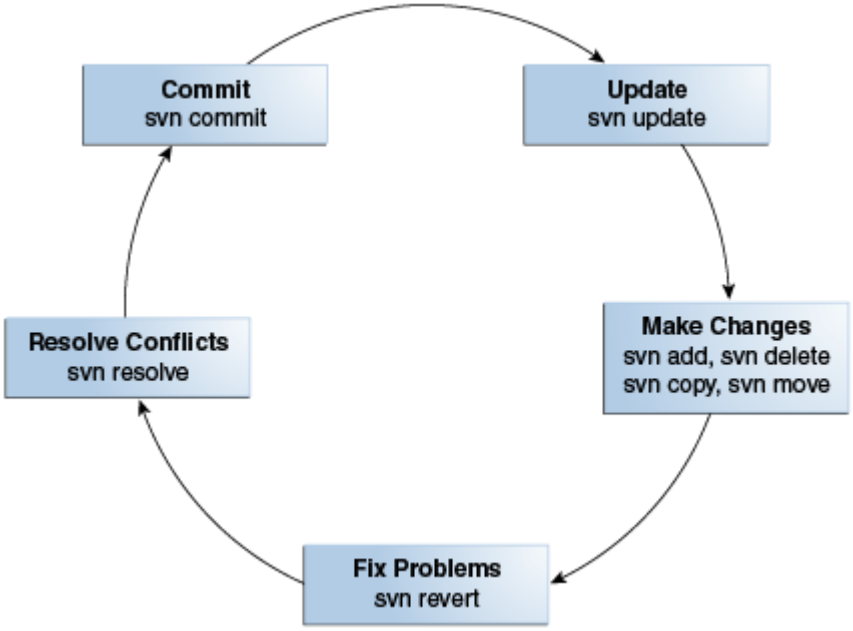
Позволяет просматривать информацию о репозитории, такую как URL родительского репозитория, номер текущей ревизии, дату последних изменений.

Пример вывода svn info:

*$ svn info*
*Path: .*
*Working Copy Root Path: /home/tkseniya/Repos/copy*
*URL: file:///home/tkseniya/Repos/parent*
*Relative URL: ^/*
*Repository Root: file:///home/tkseniya/Repos/parent*
*Repository UUID: 95d4837b-3a13-4880-a1b4-efa686ed9544*
*Revision: 0*
*Node Kind: directory*
*Schedule: normal*
*Last Changed Rev: 0*
*Last Changed Date: 2018-11-15 19:37:59 +0300 (Thu, 15 Nov 2018)*

## 2.4. Жизненный цикл проекта с использованием SVN.

Жизненный цикл (ЖЦ) можно представить следующим рисунком:




*Жизненный цикл коммита в Subversion*

После того, как вы внесли изменения в файлы вашей рабочей копии и убедились в том, что они корректно работают, Subversion предлагает вам команды «публикации» (записи в хранилище) ваших изменений, в результате чего они станут доступными для всех участников проекта. Если другие участники проекта опубликовали свои изменения, Subversion предлагает вам команды для объединения (путем чтения информации из хранилища) этих изменений с вашей рабочей копией.

*Фиксаця (check-in, commit, submit)* - создание новой версии, публикация изменений. Распространение изменений, сделанных в рабочей копии, на хранилище документов. При этом в хранилище создаётся новая версия изменённых документов.

Далее – детальный разбор каждого шага ЖЦ.

### 2.4.1. Создание репозитория / клонирование уже существующего репозитория

#### 2.4.1.1. Создание пустого репозитория

Для создания пустого репозитория необходимо выполнить команду **svnadmin create** (команда svnadmin поставляется со стандартным пакетом svn):

*$ svnadmin create /path/to/rep*

#### 2.4.1.2. Создание репозитория из исходных кодов

**svn import**

Для импортирования нового проекта в Subversion-хранилище используется *svn import*.

Команда *svn import* это быстрый способ скопировать не версионированное дерево файлов в хранилище, создавая при необходимости промежуточные директории.

*$ svnadmin create /usr/local/svn/newrepos*

*$ svn import mytree file:///usr/local/svn/newrepos/some/project *

*-m "Initial import"*

*Adding mytree/foo.c*

*Adding mytree/bar.c*

*Adding mytree/subdir*

*Adding mytree/subdir/quux.h*

*Committed revision 1.*

В предыдущем примере выполняется копирование содержимого директории mytree в директорию some/project хранилища. Просмотр файлов хранилища даст следующий результат:

*$ svn list file:///usr/local/svn/newrepos/some/project*

*bar.c*

*foo.c*

*subdir/*

Обратите внимание на то, что после завершения импорта, оригинальное дерево файлов не конвертируется в рабочую копию. Для того, чтобы начать работать вам необходимо создать новую рабочую копию (*svn checkout*) дерева файлов.

### 2.4.1.3. Клонирование репозитория

**svn checkout**

Для того чтобы создать рабочую копию уже существующего репозитория, вам нужно получить какую-либо из подкаталогов хранилища. Например, если вы получите /calc, у вас будет рабочая копия, наподобие этой:

*$ svn checkout http://svn.example.com/repos/calc*

*A calc/Makefile*

*A calc/integer.c*

*A calc/button.c*

*Checked out revision 56.*

**URL хранилища**

Получить доступ к хранилищу Subversion можно различными способами — на локальном диске или через ряд сетевых протоколов. Местоположение хранилища всегда определяется при помощи URL.

Ниже представлены виды протоколов для доступа к хранилищу:

| **Методы доступа** | **Расшифровки** |
|---|---|
| file:/// | прямой доступ к хранилищу (на локальном диске) |
| http:// | доступ через протокол WebDAV (если Subversion-сервер работает через Apache) |
| https:// | тоже что и http:// но с SSL-шифрованием |
| svn:// | доступ через собственный протокол к серверу svnserve |
| svn+ssh:// | тоже что и svn://, но через SSH-соединение |

### 2.4.2. Обновление репозитория

Когда проект скопирован другими участниками, и они внесли свои изменения, вам нужно «забрать» их к себе.

**svn update**

При командной работе над проектом обновление рабочей копии необходимо для получения любых изменений, внесённых с момента вашего последнего обновления другими разработчиками проекта. Используйте *svn update* для синхронизации вашей рабочей копии с последней правкой в хранилище.

*$ svn update*

*U button.c*

*Updated to revision 57.*

Когда сервер отправляет изменения в вашу рабочую копию для каждого элемента выводится латинская буква — код, определяющий, какое действие выполнила Subversion для приведения ваше рабочей копии в актуальное состояние.

Ниже приведена таблица с буквенными кодами, получаемыми при получении изменений с сервера:

| **Букв. обозначение** | **Расшифровка** | **Описание** |
|---|---|---|
| U | Updated | Файл был обновлён |
| A | Added | Файл был добавлен |
| D | Deleted | Файл был удалён |
| R | Replaced | Файл был перемещён |
| G | merGed | Файл был объединён |
| C | Conflictiong | Файл конфликтует с вашей локальной версией |

**Изменение хранилища без участия рабочей копии**

Существуют некоторые случаи использования, которые сразу же фиксируют в хранилище изменения структуры. Это происходит только тогда когда подкоманда оперирует напрямую с URL вместо рабочей копии. В частности, отдельные применения svn mkdir, svn copy, svn move и svn delete могут работать с URL.

Когда вы работаете напрямую с URL, любое из приведенных выше действий приводит к немедленной фиксации.

### 2.4.3. Внесение изменений в рабочую копию

Команды Subversion которыми вы будете пользоваться — это svn add, svn delete, svn copy и svn move – для изменения структуры рабочей копии, а svn add – для добавления новых файлов и директорий под контроль версий. Однако если вы просто редактируете файлы которые уже есть в Subversion ни одна из этих команд вам не нужна.

Обратите внимание, что нельзя использовать обычные команды удаления, копирования и перемещения вместо svn delete, svn copy и svn move, так как тогда ваши изменения не будут записаны в .svn и при обновлении из репозитория вы вернёте все файлы обратно.

**svn add foo**

Запланировать файл, директорию или символьную ссылку foo для добавления в Хранилище при следующей фиксации.

**svn delete foo**

Запланировать удаление из хранилища файла, директории или символьной ссылки foo. Если foo является файлом или ссылкой, он сразу же удаляется из вашей рабочей копии. Если foo является директорией, она не удаляется, но Subversion запланирует ее удаление. foo будет удалена из рабочей копии и хранилища при фиксации изменений.

**svn copy foo bar**

Создать новый элемент bar как копию foo. bar будет автоматически запланирован для добавления. Когда при следующей фиксации bar будет добавлен в хранилище в его истории будет отмечено копирование (то, что первоисточником является foo). Команда copy не создает промежуточных директорий.

**svn move foo bar**

Эта команда полностью аналогична выполнению svn copy foo bar; svn delete foo, поэтому, bar будет запланирован для добавления как копия foo, а foo будет

запланирован для удаления. svn move не создает промежуточных директорий.

**svn mkdir dir_name**

Эта команда создаёт директорию dir_name и добавляет её под контроль версий.

### 2.4.4. Исправлнеие внесённых изменений

**svn revert**

Теперь предположим, просмотрев вывод команды diff вы обнаружили, что изменения в README являются ошибочными; возможно, в своем редакторе, вы случайно набрали этот текст, предназначавшийся для другого файла.

Это как раз возможность воспользоваться svn revert.

*$ svn revert README*

*Reverted 'README'*

Subversion возвращает файл в состояние, предшествующее модификации, путём замены файла его кэшированной «первоначальной» копией из .svn-области.

Для этого Subversion использует отдельную для каждого версионированного файла кэшированную в административной области .svn первоначальную версию. Это позволяет Subversion показывать — и отменять — локальные изменения таких файлов без необходимости сетевого доступа.

### 2.4.5. Исправление конфликтов (при объединении с чужими изменениями)

Предположим вы запустили svn update получили:

*$ svn update*

*U INSTALL*

*G README*

*C bar.c*

*Updated to revision 46.*

Файл, отмеченный C имеет конфликт. Это значит, что изменения с сервера пересеклись с вашими личными и теперь вам нужно вручную сделать между ними выбор.

Всякий раз когда возникает конфликт, для того, чтобы помочь вам заметить и решить этот конфликт, возникают как правило три вещи:

1) Subversion печатает C во время обновления и запоминает, что файл в состоянии конфликта;

2) Если Subversion считает, что файл объединяемого типа она помещает маркеры конфликта — специальные текстовые строки которые отделяют

«стороны» конфликта — в файл, для того, чтобы визуально показать пересекающиеся области;

3) Для каждого конфликтного файла Subversion добавляет в рабочую копию до трех не версионированных дополнительных файлов:

- filename.mine (ваш файл в том виде в каком он был в рабочей копии до обновления (если Subversion решает, что файл не объединяемый, тогда файл .mine не создается, так как он будет идентичным рабочему файлу.)
- filename.rOLDREV (файл правки BASE, где BASE - правка которая была до обновления рабочей копии. То есть это файл который у вас был до внесения изменений)
- filename.rNEWREV (файл, который ваш Subversion-клиент только что получил с сервера, когда вы обновили рабочую копию. Этот файл соответствует правке HEAD хранилища, где HEAD — специальный указатель на последний коммит в хранилище)

Здесь OLDREV - это номер правки файла в директории .svn, а NEWREV - это номер правки HEAD хранилища.

Если вы получили конфликт, у вас есть три варианта:

- Объединить конфликтующий текст «вручную» (путем анализа и редактирования маркеров конфликта в файле).
- Скопировать один из временных файлов поверх своего рабочего файла.
- Выполнить *svn revert <filename>* для того, чтобы убрать все ваши локальные изменения.

После того, как вы решили конфликт, вам нужно поставить в известность Subversion, выполнив *svn resolved*. Эта команда удалит три временных файла и Subversion больше не будет считать, что файл находится в состоянии конфликта.

**2.4.5.1. Объединение конфликтов вручную**

Конфликтный файл будет выглядеть так:

*<<<<<<< имя файла*
*ваши изменения*
*=======*
*результат автоматического слияния с репозиторием*
*>>>>>>> ревизия*

Вам необходимо разрешить конфликты вручную редактированием данного файла или через сторонние приложения.

Когда вы выполните слияние изменений, выполните *svn resolved* и вы готовы к фиксации изменений:

*$ svn resolved sandwich.txt*

И, наконец, зафиксируйте изменения:

*$ svn commit -m "Go ahead and use my sandwich, discarding Sally's edits."*

### 2.4.5.2. Использование svn revert

Если вы получили конфликт и вместо анализа решаете отбросить изменения и начать сначала, просто отмените ваши изменения:

*$ svn revert sandwich.txt*

*Reverted 'sandwich.txt'*

Обратите внимание, что когда вы возвращаете файл к предыдущему состоянию, вам не нужно выполнять *svn resolved*.

### 2.4.6. Анализ изменений

Анализ выполняется перед тем, как зафиксировать изменения.

Цели анализа:

- составление более аккуратного лог-сообщения;
- обнаружение непреднамеренного изменения файла, что даст вам возможность до фиксации вернуть файл к предыдущему состоянию;
- пересмотр и проверка изменений перед их публикацией.

Что бы увидеть все сделанные изменения вы можете воспользоваться svn status, svn diff и svn revert. Первые две команды вы можете использовать для того, что бы найти изменённые файлы рабочей копии, а затем при помощи третьей, возможно, отменить некоторые из них.

В директории .svn рабочая копия содержит скрытую кешированую «нетронутую» копию каждого версионированного файла, что позволяет быстро откатить изменения.

**svn status**

Находит все сделанные вами файловые и структурные изменения. Ниже приведены примеры различных буквенных кодов, возвращаемых svn status.

Под объектом подразумевается файл, директория или ссылка.

Ниже приведена таблица с основными буквенными кодами команды svn status:

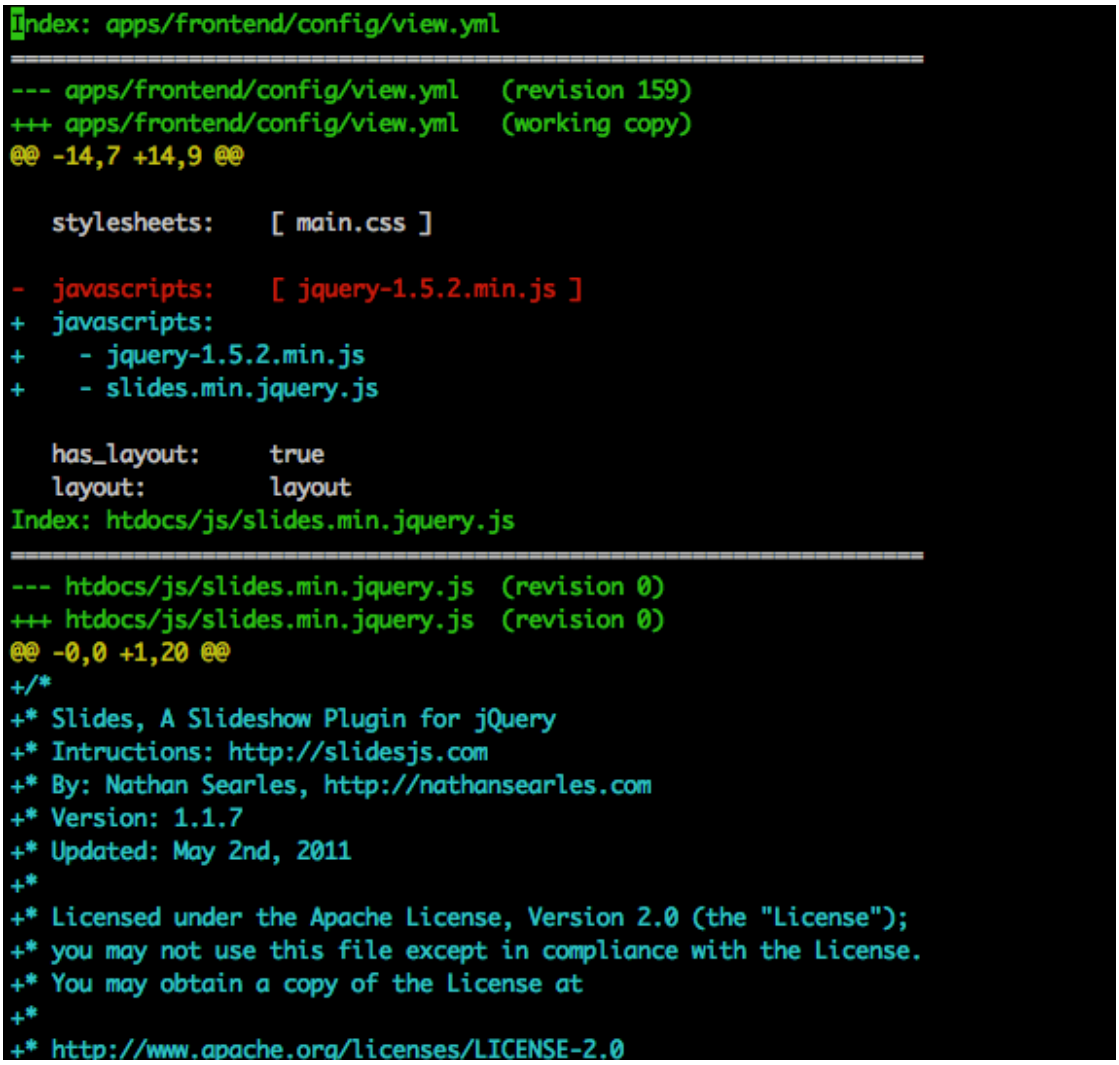
| **Буквенный код** | **Пояснение** |
|---|---|
| ' ' | Без модификаций |
| A | Объект запланирован для добавления |
| D | Объект запланирован для удаления |
| M | Объект был изменён |
| R | Объект был заменён внутри рабочей копии |

| | |
|---|---|
| C | Конфликтующий объект |
| X | Объект был включён внешне |
| I | Объект был заигнорирован |
| ? | Объект не под версионным контролем |
| ! | Объект не найден (был перемещён или удалён без использования svn move или svn delete) |
| ~ | Объект поменял свой тип (файл, директория, ссылка) |

**svn diff**

Увидеть как именно вы изменили элементы можно запустив *svn diff* без аргументов, в результате выведутся изменения файлов в виде единого формата представления различий. При этом удалённые строки предваряются -, а добавленные строки предваряются +.

Команда *svn diff* формирует свой вывод сравнивая ваши рабочие файлы с кэшированными «нетронутыми» копиями из .svn. Весь текст запланированных для добавления файлов показывается как добавленный, а весь текст запланированных для удаления файлов показывается как удалённый.




*Пример команды svn diff*

### 2.4.7. Фиксация изменений

Для фиксиации изменений в хранилище используется команда *svn commit*, она отправляет все ваши изменения в хранилище. При фиксации изменений необходимо указать описывающие ваши изменения лог сообщение. Лог сообщение будет присоединено к созданной правке. Если ваше лог сообщение короткое, вы можете указать его в командной строке, используя опцию --*message (или -m)*:

*$ svn commit --message "Corrected number of cheese slices."*

*Sending*

*sandwich.txt*

*Transmitting file data .*

*Committed revision 3.*

Однако, если вы составляли лог сообщения во время работы, можно указать Subversion взять лог сообщение из файла, передавая имя файла в параметре --file:

*$ svn commit --file logmsg*

Если вы не укажите ни опции --*message* ни опции --*file*, для составления лог сообщения, Subversion автоматически запустит редактор сообщений.

Хранилище, в целом, не знает и не заботится о смысле ваших изменений; оно только контролирует, что бы никто не изменил те же файлы что и вы. Если это все-таки случилось вся фиксация будет отклонена с сообщением информирующим вас, что один или несколько файлов устарели:

*$ svn commit --message "Add another rule"*

*Sending*

*rules.txt*

*svn: Commit failed (details follow):*

*svn: Out of date: 'rules.txt' in transaction 'g'*

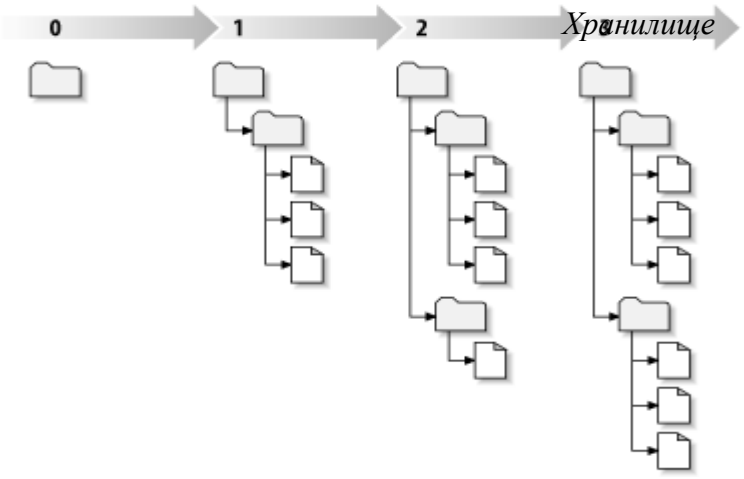
Теперь вам нужно выполнить svn update разобраться со всеми объединениями и конфликтами и попытаться выполнить фиксацию снова.

Операция svn commit может опубликовать изменения любого количества файлов и каталогов за одну атомарную операцию (либо изменения вносятся полностью, либо не вносятся вообще). В своей рабочей копии вы можете менять содержимое файлов, создавать, удалять, переименовывать и копировать файлы и каталоги, а затем зафиксировать все изменения за один раз.

Каждый раз, когда происходит фиксация, создаётся новое состояние файловой системы, которое называется *правка*. Каждая правка получает

уникальный номер, на единицу больший номера предыдущей правки. Начальная правка только что созданного хранилища получает номер 0 и не содержит ничего, кроме пустого корневого каталога.

Представьте массив номеров правок, начинающийся с 0, с направлением слева направо. Каждый номер правки имеет соответствующее дерево файлов, а каждое дерево представляет собой «снимок» того, как хранилище выглядело после фиксации.




*Пример ревизий хранилища*

## 2.5. Просмотр истории изменений

**svn log**

Показывает вам развернутую информацию: лог сообщения, присоединенные к правкам, с указаной датой изменений и их автором, а также измененные в правке пути файлов.

*$ svn log*

---

*r3 | sally | Mon, 15 Jul 2002 18:03:46 -0500 | 1 line*
*Added include lines and corrected # of cheese slices.*
*---*
*r2 | harry | Mon, 15 Jul 2002 17:47:57 -0500 | 1 line*
*Added main() methods.*
*---*
*r1 | sally | Mon, 15 Jul 2002 17:40:08 -0500 | 1 line*
*Initial import*

Обратите внимание на то, что по умолчанию лог сообщения выводятся в обратном хронологическом порядке. Если вам нужно увидеть другой диапазон правок в заранее определенном порядке или только одну правку, укажите параметр *--revision (-r)*:

*$ svn log --revision 5:19* – показывает ревизии с 5 под 19 в хронологическом порядке.

**svn cat**

Если вы хотите проанализировать ранние версии файла, а не различия между двумя файлами, можно воспользоваться svn cat:

*$ svn cat --revision 2 rules.txt*

**svn list**

Команда *svn list* показывает содержимое директории в хранилище, при этом не закачивая его на локальную машину:

Дополнительно ко всем упомянутым выше командам, можно воспользоваться svn update и svn checkout с параметром --revision для того, что бы переместить рабочую копию «назад во времени» (todo рассказать что это полезно)

*$ svn update --revision PREV foo.c*
*# отменить последние изменения в foo.c*
*# (рабочая правка foo.c понижается)*

## 2.6. Другие полезные команды

**svn cleanup**

Когда Subversion изменяет рабочую копию (или любую информацию в области .svn) она пытается делать это как можно более безопасно. Перед изменением рабочей копии Subversion записывает свои намерения в лог файл. Затем для выполнения запрошенных изменений она выполняет команды из лог файла, устанавливая блокировку той части рабочей копии, с которой работает — делается это для невозможности работы других Subversion-клиентов с

рабочей копией, которая находится в промежуточном состоянии. После выполнения запрошеных действий Subversion удаляет лог файл. Архитектурно, это напоминает журналируемую файловую систему. Если работа Subversion была прервана (в результате того, что процесс был убит или, например, из-за машинного сбоя) лог файлы остаются на диске.

Перезапустив выполнение лог файлов, Subversion может завершить предварительно начатые операции и рабочая копия снова вернется в согласованное состояние.

*svn cleanup* выполняет поиск и выполнение незавершенных лог файлов, удаляя по ходу выполнения блокировки в рабочей копии. Если Subversion когда-нибудь говорила вам о том, что часть рабочей копии «заблокирована» тогда вам нужно запустить эту команду.

**Ключевые слова правок**

В каждом каталоге рабочей копии есть служебный подкаталог .svn. Для каждого файла в каталоге Subversion сохраняет копию этого файла в служебной папке. Эта копия является немодифицированной копией файла какой он есть в последней правке (называемой «BASE») до которой вы обновили его в вашей рабочей копии.

*HEAD*

Последняя (или «самая новая») правка хранилища

*BASE*

Номер правки элемента рабочей копии. Если элемент редактировался, то «BASE версия» соответствует тому, как элемент выглядел до редактирования.

*COMMITTED*

Правка, в которой элемент последний раз редактировался (предшествующая либо равная BASE).

*PREV*

Правка, предшествующая последней правке, в которой элемент был изменен.

**Замечание**

*PREV, BASE* и *COMMITTED* могут использоваться при обращении по локальным путям, но не по URL.

Ниже приведено несколько примеров использования ключевых слов правок:

*$ svn diff --revision PREV:COMMITTED foo.c*

*# показать последнее изменение зафиксированное для foo.c*

*$ svn log --revision HEAD*

*# показать лог для последней фиксации хранилища*

*$ svn diff --revision HEAD*

*# сравнить ваш рабочый файл (с учетом локальных изменений)*

*# с последней правкой в хранилище*

*$ svn diff --revision BASE:HEAD foo.c*

*# сравнить ваш «исходный» foo.c (без учета локальных*

*# изменений) с последней версией в хранилище*

*$ svn log --revision BASE:HEAD*

*# показать все логи фиксаций со времени вашего последнего обновления*

Эти ключевые слова позволят вам выполнять многие часто используемые операции без необходимости обращаться к конкретным номерам правок или точно помнить номер правке своей рабочей копии.

**Игнорирование файлов**

*Игнорирование* — процесс, позволяющий хранить файлы в репозитории не под версионным контролем. Subversion не предлагает добавить заигнорированные файлы в следующий коммит.

Игнорирование является очень полезной командой, на практике вы будете пользоваться ей очень часто.

Например, у вас есть проект на c++, и файлы, которые необходимо хранить — это исходные коды формата .cpp и .hpp, .c и .h. Но необходимо добавить в игнорированные файлы .o и .a, т.к. они получаются при компиляции. Для каждой модели архитектуры эти файлы будут разными. Другой человек, склонировавший ваш проект, должен скомпилировать их сам.

В svn существует несколько типов игнорирования:

- Глобальное игнорирование;
- Локальное игнорирование внутри проекта.

Глобальное игнорирование необходимо для остановки отслеживания определённых файлов во всех репозиториях на данном компьютере. Оно никак не отражается на репозиториях других пользователей или на центральном хранилище.

Например, для пользователей MacOS, можно добавить в игнорируемые скрытый файл .DS_Store, задающий настройки папки (расположение на экране и т.д.), т.к. для каждого пользователя MacOS, эти файлы будут свои, и ваши настройки никому не нужны, а для пользователей других ОС вообще бесполезны.

Локальное игнорирование действует внутри репозитория и необходимо, чтобы определённый файлы были не видны всем его пользователям (пример с исходными файлами на c++ выше).

Например, локальное игнорирование файлов с расширением jpg:

*svn propset svn:ignore "*.jpg" .*

Для просмотра игнорированных файлов, необходима команда svn status с ключом —no-ignore.

*svn status --no-ignore | grep "^I"*

Вы получите следующий результат:

*I myimage.jpg*

**Метки**

Другой полезной операцией является проставления меток. Можете считать, что метки — это специальные обозначения, прикреплённые к коммиту. С помощью них удобно искать коммиты, в которых приложение было в определённом состоянии (например, релиз 1.0, релиз 1.2).

Для меток в Subversion нет специальной команды, они выглядят как дешевые копии (cheap copies), или ссылки):

*svn copy http://svn.example.com/project/trunk *

*http://svn.example.com/project/tags/1.0 -m "Release 1.0"*

**Использованные источники:**


- https://sobek.su/Docs/svn-book/svn.basic.vsn-models.html
- http://svnbook.red-bean.com/en/l1.8/svn.ref.svn.c.status.htm