Системы контроля версий (от англ. Version Control System, VCS)

При разработке более-менее крупных проектов мы можем столкнуться с рядом трудностей, например:

1. Изменяем код, а потом хотим откатить изменения. Каждую версию можно сохранять в отдельную папку, но со временем станет сложно управляться с большим количеством файлов.
2. Если над программой работает сразу несколько человек, было бы неплохо автоматизировать процесс объединения сделанных ими изменений.

Для решения этих проблем удобно использовать системы контроля версий, например Subversion (SVN).

Обычно VCS состоит из двух частей:

- **Сервер**, или репозиторий — где хранятся все исходные коды программы, а также история их изменения.
- **Клиент**. Каждый клиент имеет свою локальную копию (working copy) исходных кодов, с которой работает разработчик.

В связи с тем, что разработчики работают только с локальными копиями, могут возникать трудности, когда два и более человек изменяют один и тот же файл.

В VCS есть две модели, которые позволяют избегать этой проблемы:

- **Блокировка — изменение — разблокировка.**
  Согласно этой модели, когда кто-либо начинает работу с файлом, этот файл блокируется, и все остальные пользователи теряют возможность его редактирования.
  Очевидным недостатком такой модели является то, что файлы могут оказаться надолго заблокированными, что приводит к простоям в разработке проекта.
- **Копирование — изменение — слияние.**
  В данной модели каждый разработчик свободно редактирует свою локальную копию файлов, после чего выполняется слияние изменений.
  Недостаток этой модели в том, что может возникать необходимость разрешения конфликтов между изменениями файла.

В SVN доступны обе модели, причём вторая является основной. В то же время для некоторых типов файлов (например, изображения) целесообразно использовать первую.

Subversion (SVN)

SVN является одной из клиент-серверных систем контроля версий и состоит из двух частей:

- Svnserve — серверная часть
  В качестве серверной части можно использовать обычный http-сервер.
- svn — клиентская часть

Можно создать и настроить собственный SVN-сервер или использовать готовые бесплатные сервисы: code.google.com , sourceforge.net .

Команды SVN

Перед тем как приступить к использованию SVN, ознакомимся с некоторыми командами, которые может выполнять клиентская часть.

Наиболее часто используемые разработчиками команды:

- **svn update** — обновляет содержимое локальной копии до самой последней версии из репозитория.
- **svn commit** — отправляет все изменения локальной копии в репозиторий.
- **svn add <*файл/папка*>** — включить файл/папку в локальную копию проекта

Нередко также используются команды:

- **svn move <*файл/папка1*> <*папка2*>** — переместить файл/папку1 в папку2
- **svn copy <*файл/папка*> <*папка*>** — скопировать файл/папку1 в папку2
- **svn delete <*файл/папка*>** — удалить файл/папку из локальной копии проекта

Прочие полезные команды SVN:

- **svn list <*URL*>** — просмотр каталога репозитория
- **svn log <*файл*> --verbose** — история изменения файла по ревизиям
- **svn cat --revision <*номер_ревизии*> <*файл*>** — отображение содержимого файла из данной ревизии
- **svn diff --revision <*номер_ревизии1:номер_ревизии2*> <*файл*>** — отображение изменений файла между двумя ревизиями
- **svn status** — отображение изменений в локальной копии относительно репозитория

Жизненный цикл проекта на SVN

Допустим, что у нас уже есть настроенный SVN-сервер, и мы хотим перенести на него проект. У нас есть папка test, в которой лежит три файла:

1.cpp
2.cpp
Makefile

0) Добавляем новые файлы на сервер:

```
svn import svn://......../repo/test test
```

Здесь *svn://......../repo/test* - url-адрес проекта, *test* - название добавляемой папки.

1) Получаем локальную копию файлов проекта:

```
svn checkout svn://......../repo/test project/test
```

project/test - адрес, где будет располагаться локальная копия.

После выполнения этой команды получим следующую структуру файлов:
project
  test
    .svn
    1.cpp
    2.cpp
    Makefile

Локальную копию не следует добавлять в ту папку, из которой делали import, так как при совпадении имён файлов они не будут перезаписаны.

2) Изменяем файлы

Допустим, на этом шаге мы хотим сделать какие-нибудь изменения в проекте, а именно:

- изменить содержимое 1.cpp следующим образом:
  до изменений:
  ```
  int main(){
    return 0;
  }
  ```

  после изменений:
  ```
  int main(){
    printf(" ");
    return 0;
  }
  ```

- удалить файл 2.cpp

Стоит обратить внимание на следующую особенность SVN:
содержимое файлов проекта можно менять в любых привычных редакторах, **но** изменения в структуре файлов проекта следует выполнять с помощью соответствующих команд svn (add, copy, move, delete).
То есть если мы просто удалим файл из папки с локальной копией - это никак не отразится на содержимом репозитория даже после успешного commit'a.

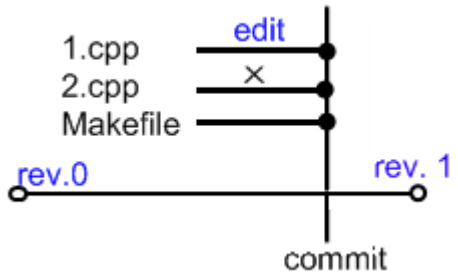
3) Фиксируем состояние

При фиксации происходит отправка всех изменений в локальной копии на сервер.

```
svn commit -m "....."
```
-m "....." — комментарий к commit'у.
При выполнении этой команды SVN узнаёт нужную ему информацию о проекте из папки .svn

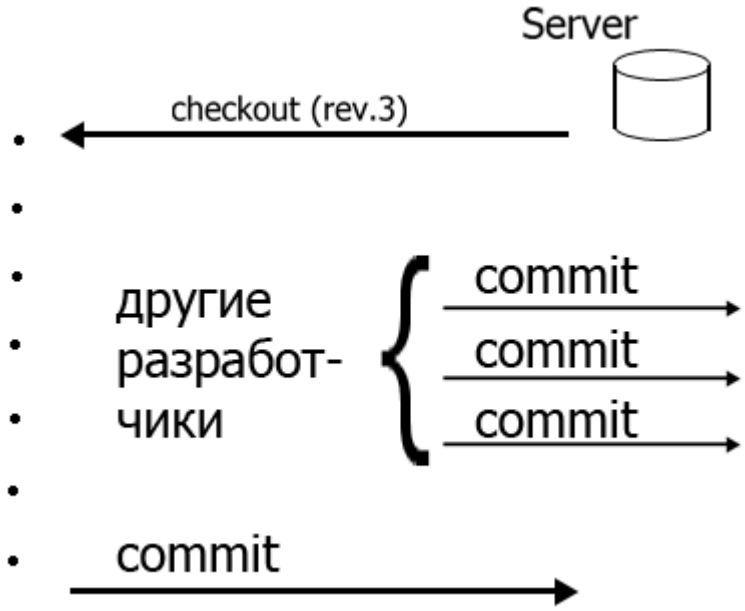
После каждого commit'a номер ревизии увеличивается на единицу.
Номер ревизии (Revision) &mdash целое число, показывающее номер состояния проекта.

Все выполненные действия можно схематически представить на рисунке:



Разрешение конфликтов

При работе над проектом возможны такие ситуации:



Допустим, в какой-то момент времени разработчик обновляется до последней версии проекта и начинает работу над некоторыми файлами.
Через какое-то время, изменив файлы, он пытается сделать commit. Однако, этот commit не удастся, если другие разработчики уже вносили изменения в те же файлы проекта и сохраняли их в репозитории.
Такая ситуация называется конфликтом. У разработчика, чей commit не удался из-за конфликтов, есть два возможных варианта действий:

1. **svn revert** — отменить все свои изменения
2. **svn update** — забрать из репозитория новые версии файлов и, разрешив конфликты, снова попытаться сделать commit.

Как разрешаются конфликты

Допустим, в репозитории хранится файл **main.cpp** ревизии 13:

```
int main(){
  fprintf();
  return 0;
}
```

В какой-то момент времени user1 и user2 одновременно делают update:	`svn update`	`svn update`

Оба разработчика вносят изменения в main.cpp	`int main(){`
	`  fscanf();`
	`  return 0;`
	`}`
		`int main(){`
		`  fpintfpr();`
		`  return 0;`
		`}`

Затем user2 commit'ит свои изменения		`svn commit`
commit user1 не удастся, так как есть конфликт	`svn commit`	
Поэтому ему приходится делать update до последней версии	`svn update`	

Теперь у user1 есть несколько путей разрешения конфликта:

- убрать чужие изменения, оставив свои
- не вносить свои изменения, оставив всё как есть
- вручную соединить все изменения в одном файле

В последнем случае user1 имеет несколько файлов:
main.cpp — этот файл уйдёт в репозиторий после разрешений конфликта
main.cpp.mine — в этом файле хранятся изменения, сделанные user1
main.cpp.r13 — начальная версия файла без всяких изменений
main.cpp.r14 — файл, попавший в репозиторий (с изменениями user2)

После сведения всех изменений в **main.cpp** user1 пишет
`svn resolved main.cpp` (при этом удалятся временные файлы)
`svn commit`

SVN позволяет так же объединить любую версию с любым набором других версий с помощью `update` и `merge`.
Про команду merge можно прочитать, набрав в консоли команду `svn help merge`.

Заключение

В этой лекции мы рассмотрели систему контроля версий SVN, которая является централизованной (то есть у любого проекта есть одно хранилище).
В настоящее время так же распространены распределённые системы управления версиями (англ. Distributed Version Control System, **DVCS**), в которых вся история изменений файлов хранится в локальных копиях клиентов и, при необходимости, синхронизируется.
К DVCS относятся: git, Mercurial, Bazaar.