Контроль версий

Лекция 12

Проблема контроля версий

- Проблема контроля версий является одной из фундаментальных проблем программной инженерии в силу :
  - невозможности практической реализации линейной стратегии разработки программных средств;
  - коллективного характера процесса разработки

Проблема контроля версий

- Даже при индивидуальной работе над проектом разработчик вынужден хранить две и более версий системы, чтобы иметь возможность вернуться на предыдущие стадии разработки
- «Ручная» работа с несколькими версиями весьма утомительна и непродуктивна

Проблема контроля версий

- При коллективной разработке ситуация существенно усложняется, поскольку возникают проблемы:
  - семантической совместимости фрагментов программного кода, созданных отдельными разработчиками;
  - оповещения об изменениях, вносимых в ранее написанный код каждым из разработчиков

Актуальность проблемы

- Особую актуальность проблема контроля версий приобрела с появлением инструментальных средств быстрой разработки проектов таких, как Delphi, C++ Builder и JBuilder, а затем Eclipse, Visual Studio.Net
- Применение этих инструментальных средств привело к взрывообразному росту производительности труда разработчиков

Актуальность проблемы

- Следствием этого стала перегрузка отдельных разработчиков и их групп проектами, исходным кодом и документацией
- Стала очевидной необходимость пересмотра подходов к организации процесса создания ПС
- Одним из аспектов нового подхода к организации разработки стало использование *систем управления версиями* – Version Control Systems (VCS)

Определение VCS

- *Системой управления версиями* называется комплекс программного обеспечения, назначением которого является централизованное хранение и обработка всех или большей части файлов, из которых состоит проект некоего программного продукта
- Основным назначением системы управления версиями является поддержка работы с изменяющейся информацией

Репозиторий VCS

- Важнейшей частью любой системы контроля версий является *репозиторий – хранилище данных*, с которыми ведется работа
- Как правило, эти данные хранятся в репозиториях в виде файлов
- В дальнейшем наряду с термином «репозиторий» мы будем использовать его русский эквивалент «хранилище»

Определение проекта

- Под *проектом* понимается совокупность файлов, включающая:
  - исходные тексты на различных языках программирования;
  - исполняемые, ресурсные и библиотечные файлы, необходимые для сборки программного продукта;
  - исходные тексты файлов справки;
  - сценарии программ инсталляции;
  - сопроводительная документация проекта
- В репозитории VCS могут храниться данные из нескольких проектов

Понятие версии

- Понятие «версия» в разных системах трактуется двумя различными способами
- Одни системы (например, CVS – *Concurent Version System*) поддерживают версионность *файлов*; это означает, что любой файл, появляющийся в проекте, получает собственный номер версии (обычно — номер 1, условной «нулевой» версией файла считается пустой файл с тем же именем)

Версии файлов

- При каждой фиксации разработчиком изменений, затрагивающих файл, файл получает новый, обычно следующий по порядку, номер версии
- Поскольку фиксации обычно затрагивают только часть файлов в репозитории, номера версий файлов, со временем расходятся, и проект в целом никакого «номера версии» не имеет, т.к. состоит из множества файлов с различными номерами версий

Версия репозитория

- Для других систем понятие «версия» относится не к отдельному файлу, а к репозиторию целиком
- Вновь созданный пустой репозиторий имеет версию 1 или 0, любая фиксация изменений приводит к увеличению этого номера (то есть даже при изменении одного файла на один байт весь репозиторий считается изменённым и получает новый номер версии)

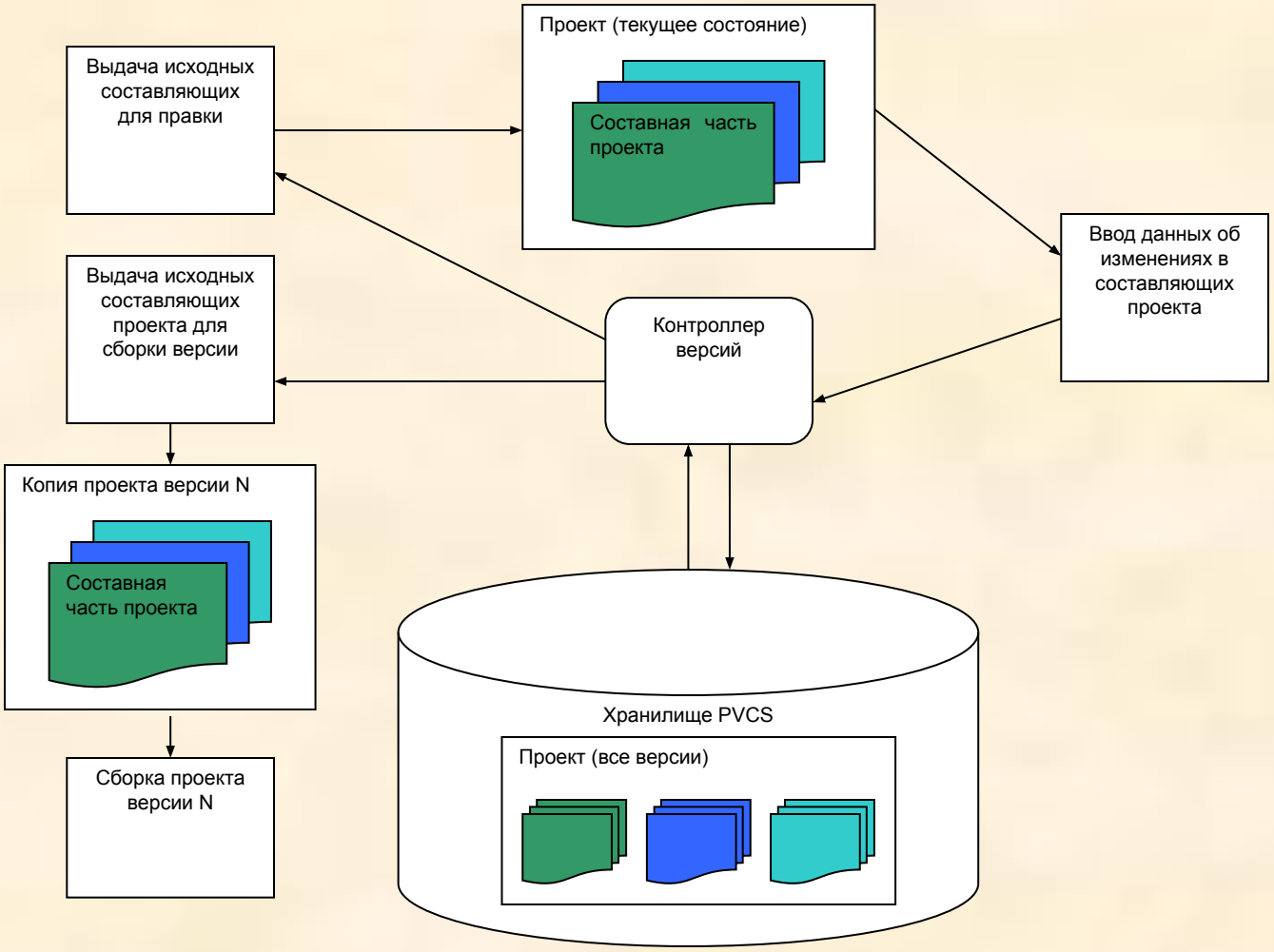
Версия репозитория

- Номера версии отдельного файла здесь, фактически, не существует
- Поэтому применительно к таким системам контроля версий под версией файла понимается та версия репозитория, в которой этот файл был изменён в последний раз

Контроль версий проекта

- Для практических целей обычно имеет значение не отдельный файл и не репозиторий в целом, а весь проект целиком
- Системы контроля версий, обеспечивающие совместное хранение данных для каждого проекта, называются *системами контроля версий проекта* (PVCS – *Project Version Control System*)

Схема контроля версий проекта



Основные процессы VCS

- Основными процессами в эксплуатации системы управления версиями являются:
  - ввод первичной информации о проекте, создание первой копии проекта в хранилище;
  - извлечение из хранилища отдельных составляющих проекта *(check-out, clone)* и создание рабочей копии для внесения изменений;
  - синхронизация рабочей копии до некоторого заданного состояния хранилища (*update, sync, switch*);

Основные процессы

- создание новой версии *(check-in, commit, submit)* с фиксацией изменений;
- извлечение из хранилища всех составляющих проекта *(pull)* заданной версии для сборки программного продукта или отдельного его компонента;
- запись в хранилище *(push)* модифицированных или вновь созданных составляющих проекта с присвоением номера версии;

Рабочая копия

- Рабочая копия – это моментальный «снимок» состояния хранилища или некоторой его части, сохраненный на компьютере клиента
- В простейшем случае это может быть копией отдельного документа, но может представлять собой копию проекта, или даже хранилища, в целом

Конфликтная ситуация

- Коллективная работа над проектом требует наличия специальных средств поддержки и обеспечения безопасности данных
- В частности, типичной в таких случаях является конфликтная ситуация, когда два или более разработчиков параллельно редактируют один и тот же файл, внося в него те или иные изменения

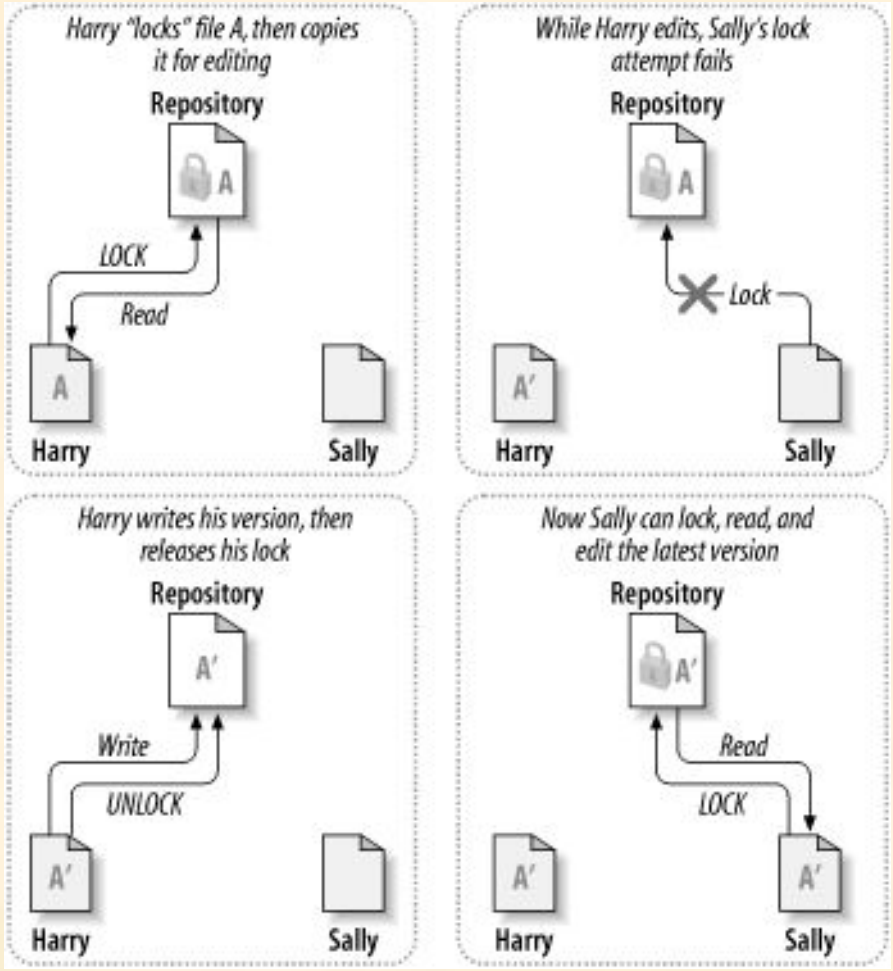
Модели версионирования

- Для решения проблемы конфликтных ситуаций используют те или иные *модели версионирования*
- Рассмотрим две основные модели:
  - модель «Блокирование – Изменение – Разблокирование»;
  - модель «Копирование – Изменение – Слияние»

Модель Блокирование-Изменение-Разблокирование

- Эта модель запрещает одновременное редактирование файла несколькими клиентами
- Перед началом редактирования клиент должен заблокировать файл
- Тогда доступ к этому файлу других клиентов станет возможен только после снятия блокировки

Пример работы модели



Недостатки модели

- Требует повышенного внимания службы администрирования ввиду возможности сохранения блокировки при некорректном завершении клиентом редактирования файла
- Приводит к замедлению процесса разработки из-за частых блокировок, выполняемых и в случаях, когда в этом нет необходимости
- Блокирование может вызвать ложное чувство безопасности в ситуации, когда два клиента редактируют разные, но *зависящие друг от друга* файлы

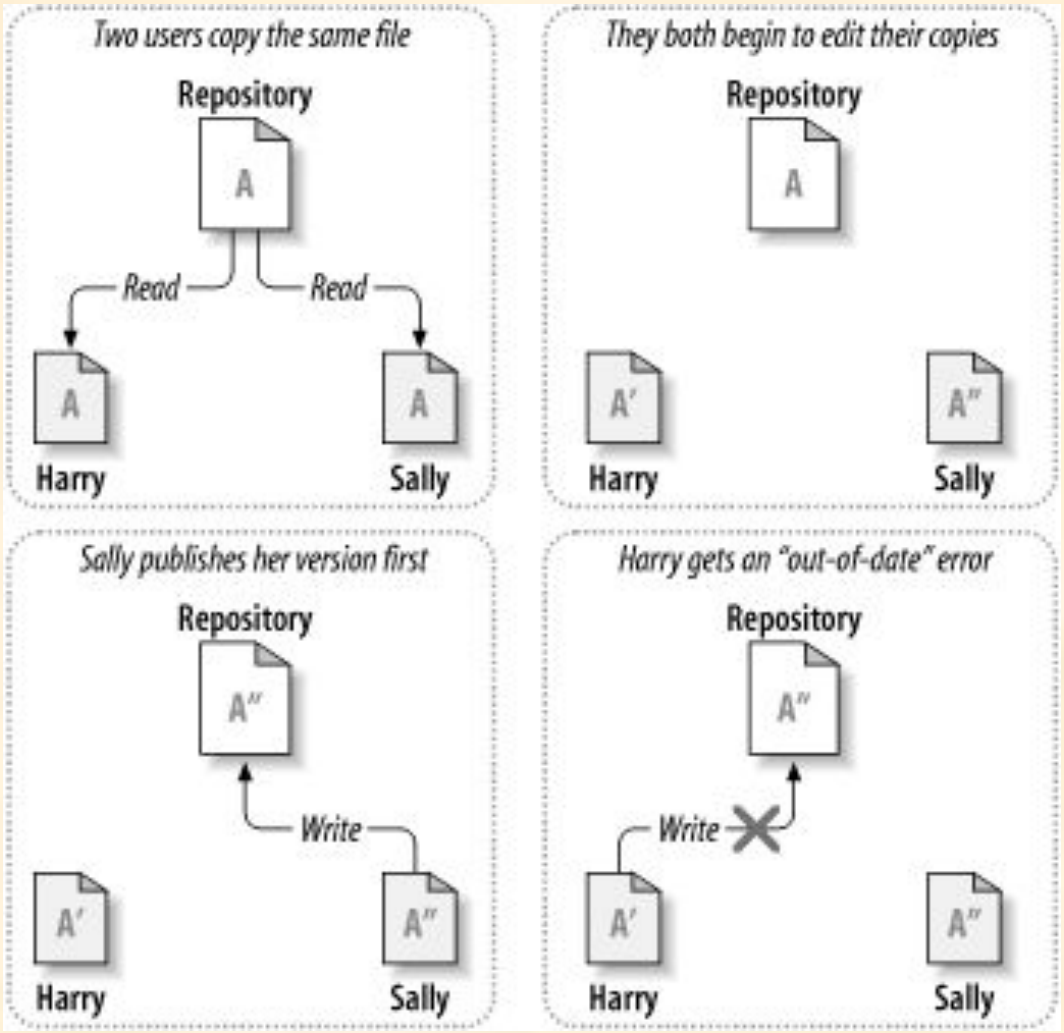
Модель Копирование-Изменение-Слияние

- Каждый клиент связывается с хранилищем проекта и создает персональную *рабочую копию* - локальное отражение файлов и каталогов хранилища
- После этого клиенты работают одновременно и независимо друг от друга, изменяя свои личные копии

Модель Копирование-Изменение-Слияние

- По завершении редактирования личные копии сливаются в новую, итоговую версию
- Обычно система управления версиями помогает в слиянии, но за его корректное выполнение отвечает человек

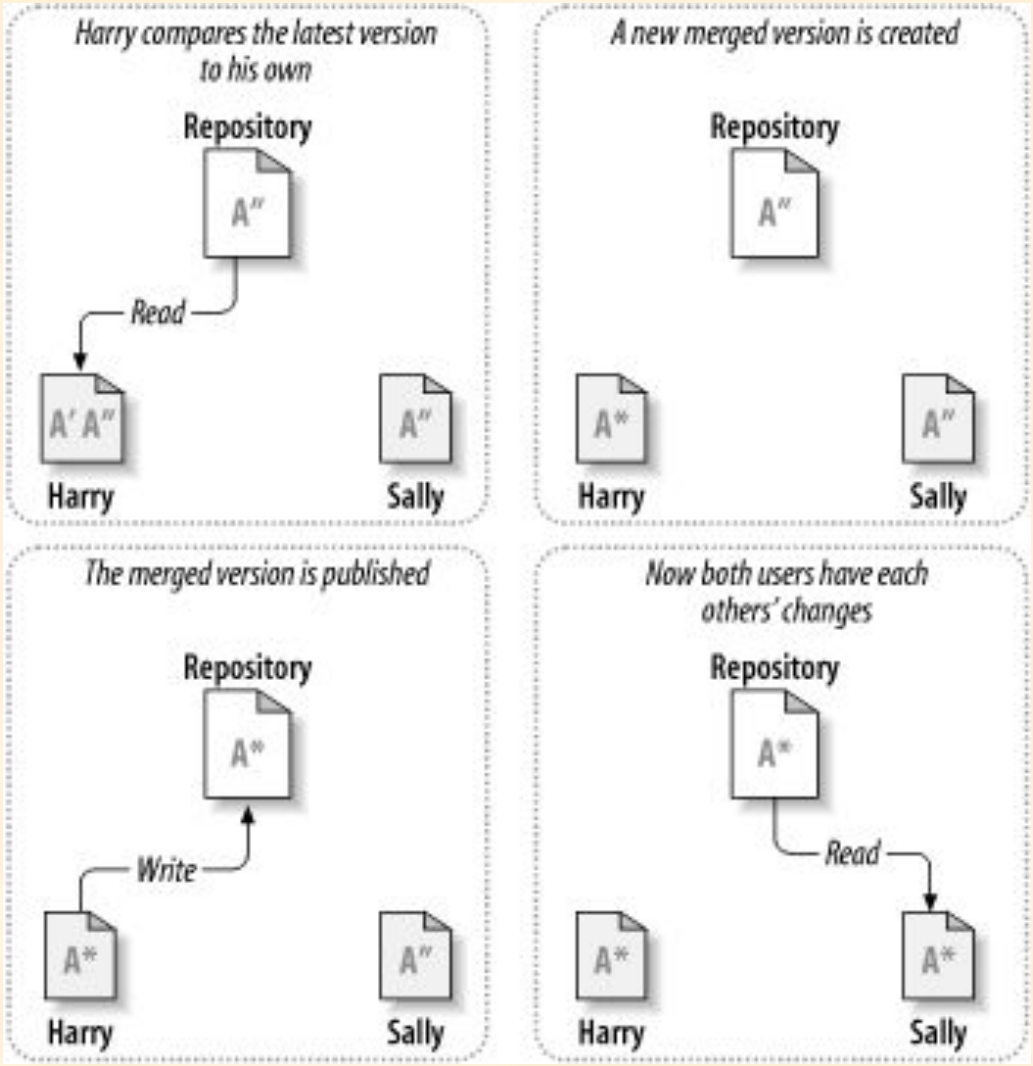
Пример работы модели



Пример работы модели

- Пользователь Гарри попытался записать в хранилище исправленный им файл A после того, как пользователь Салли уже зафиксировала там свои итоги редактирования этого файла
- Система сообщила Гарри, что его файл *устарел*

Пример работы модели



Пример работы модели

- Гарри попросил систему обновить его копию файла A
- Измененный Салли файл A копируется в рабочее пространство Гарри
- Создается обновленный файл A с включением изменений, сделанных каждым из пользователей

Объединение изменений

- При создании обновленного файла могут возникнуть две ситуации:
  - нормальная – изменения, внесенные различными разработчиками, не перекрываются; их объединение происходит в автоматическом режиме;
  - конфликтная – некоторые из внесенных изменений перекрываются; объединение таких изменений производится «вручную» после взаимных консультаций разработчиков

Способы сохранения изменений

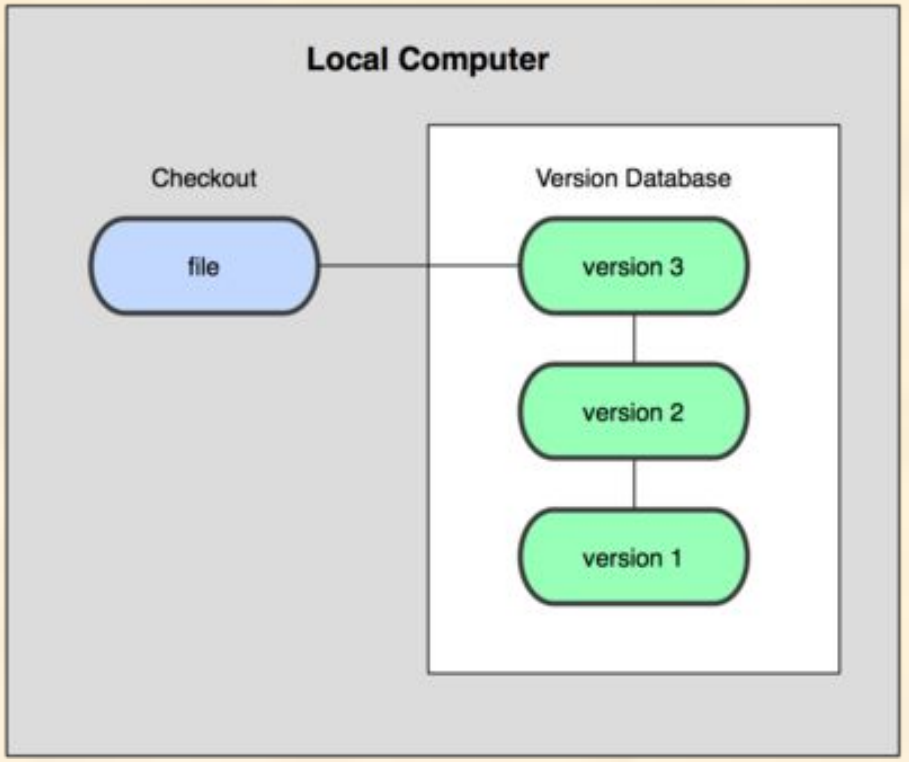
- Сохранение внесенных изменений может производиться двумя способами:
  - созданием новой, измененной копии файла;
  - сохранением только отличий новой версии от предыдущей (*дельта-компрессия*)
- Очевидно, что в последнем случае достигается значительная экономия дискового пространства

В настоящее время существует большое число различных продуктов, предлагающих широкие возможности для управления версиями. Далее будут рассмотрены лишь часть из наиболее популярных систем

ОБЗОР СИСТЕМ КОНТРОЛЯ ВЕРСИЙ

Локальные системы контроля версий

- Этот вид систем контроля версий ориентирован на проекты, ведомые одним разработчиком или небольшим коллективом, и основан на концепции версии файла



Система контроля изменений

- Примером локальной системы контроля версий является система контроля изменений
- *Система контроля изменений* (Revision Control System, RCS) была разработана в 1985 году Вальтером Тичи (Walter F. Tichy).
- Для текстовых файлов в RCS сохраняются не все версии файла, а только последняя версия и все внесенные в нее изменения (дельта-компрессия)

Система контроля изменений

- RCS также может отслеживать изменения в бинарных файлах, но при этом каждое изменение хранится в виде отдельной версии файла.
- Для обеспечения возможности коллективной работы используется механизм блокировок (модель версионирования «Блокирование – Изменение – Разблокирование»)

Достоинства

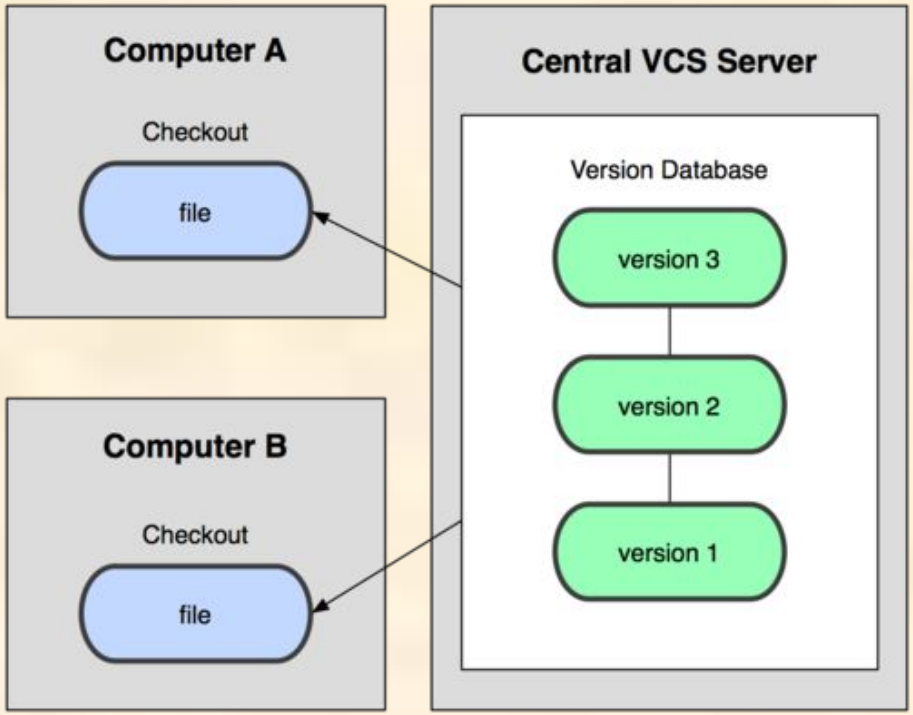
- RCS проста в использовании и хорошо подходит для ознакомления с принципами работы систем контроля версий
- Удобна для резервного копирования отдельных файлов, не требующих частого изменения группой пользователей
- Широко распространена и предустановлена почти во все системы UNIX-системах

Недостатки

- Отслеживает изменения только отдельных файлов, что не позволяет использовать ее для управления версиями больших проектов
- Не позволяет одновременно вносить изменения в один и тот же файл нескольким пользователям
- Имеет низкую функциональность по сравнению с современными системами контроля версий

Централизованные системы контроля версий

- Централизованные системы используют единый репозиторий для хранения версий различных проектов
- Репозиторий размещается на сервере и клиенты имеют возможность получить с него локальную версию проекта



Система конкурирующих версий

- Продолжением проекта RCS стала *система конкурирующих версий* (Concurrent Versions System – CVS), разработанная Диком Груном (Dick Grune) в середине 1980-х годов
- CVS использует архитектуру клиент-сервер, в которой вся информация о версиях хранится на локальном или сетевом сервере

Система конкурирующих версий

- Помимо обработки индивидуальных файлов CVS позволяет управлять группами файлов расположенных в директориях
- CVS также позволяет вести несколько линий разработки проекта с помощью ветвей (branches) разработки
- Таким образом, можно исправлять ошибки в очередной версии проекта и параллельно разрабатывать новую функциональность

Система конкурирующих версий

- В чистом виде CVS и ее клоны являются системами командной строки, поэтому для их комфортного использования необходима графическая оболочка
- Для Windows в качестве такой оболочки может использоваться продукт WinCVS, распространяемый с открытым исходным кодом

Достоинства

- Обеспечивает возможность коллективной работы над проектом
- Позволяет управлять не одним файлом, а целыми проектами
- Обладает большим количеством удобных графических интерфейсов
- Предустановлена в большинстве операционных систем семейства Linux

Недостатки

- При перемещении или переименовании файла или директории теряются все, привязанные к этому файлу или директории, изменения
- Сложности при ведении нескольких параллельных веток одного и того же проекта.
- Для каждого изменения бинарного (не текстового!) файла сохраняется вся версия файла, а не только внесенное изменение

Недостатки

- С клиента на сервер измененный файл всегда передается полностью,
- Ресурсоемкость операции, так как они требуют частого обращения к репозиторию, и избыточность сохраняемых копий

Система Subversion

- Еще одним примером централизованной системы управления версиями является система Subversion
- Subversion (SVN) была разработана в 2000 году по инициативе фирмы CollabNet и по сравнению с CVS:
  - обеспечивает лучшее управление изменениями структуры каталогов;
  - более эффективно хранит двоичные файлы;
  - имеет стандартную возможность возврата к состоянию проекта на определенный момент времени

Принцип работы

- Клиенты копируют файлы из хранилища (репозитория), создавая локальные *рабочие копии*, затем вносят изменения в рабочие копии и публикуют эти изменения в хранилище
- При этом несколько клиентов могут одновременно обращаться к хранилищу

Модели версионирования

- Для совместной работы над файлами в Subversion преимущественно используется модель «Копирование – Изменение – Слияние»
- Кроме того, для файлов, не допускающих слияния (различные бинарные форматы файлов), можно использовать модель «Блокирование – Изменение – Разблокирование»

Модель сохранения изменений

- При сохранении новых версий всех файлов используется *дельта-компрессия*: система находит отличия новой версии от предыдущей и записывает только их, избегая дублирования данных

Рабочие копии в Subversion

- Рабочая копия в Subversion представляет собой обычное дерево каталогов, содержащее набор различных файлов
- Файлы рабочей копии могут произвольным образом редактироваться разработчиком, оставаясь недоступными другим разработчикам

Рабочие копии

- После внесения изменений в файлы рабочей копии и проверки их корректности разработчик может записать свою версию в хранилище, т.е. *опубликовать*
- Если другие участники проекта производили редактирование тех же файлов и уже опубликовали свои изменения, Subversion предоставляет возможность для объединения этих изменений с рабочей копией данного разработчика

Служебный каталог

- Рабочая копия содержит дополнительные файлы, созданные и обслуживаемые Subversion, которые используются при выполнении слияний
- В частности, каждый каталог рабочей копии содержит подкаталог с именем *.svn* который называется *служебным каталогом* рабочей копии
- Файлы в служебном каталоге помогают определить какие файлы рабочей копии содержат неопубликованные изменения, и какие файлы устарели по отношению к файлам других участников

Хранилище

- Представляет собой последовательность фиксированных состояний размещенной в ней файловой системы
- Хранилище создается с помощью входящей в состав поставки Subversion утилиты SvnAdmin путем выполнения команды:

```
svnadmin create <путь к репозиторию>
```

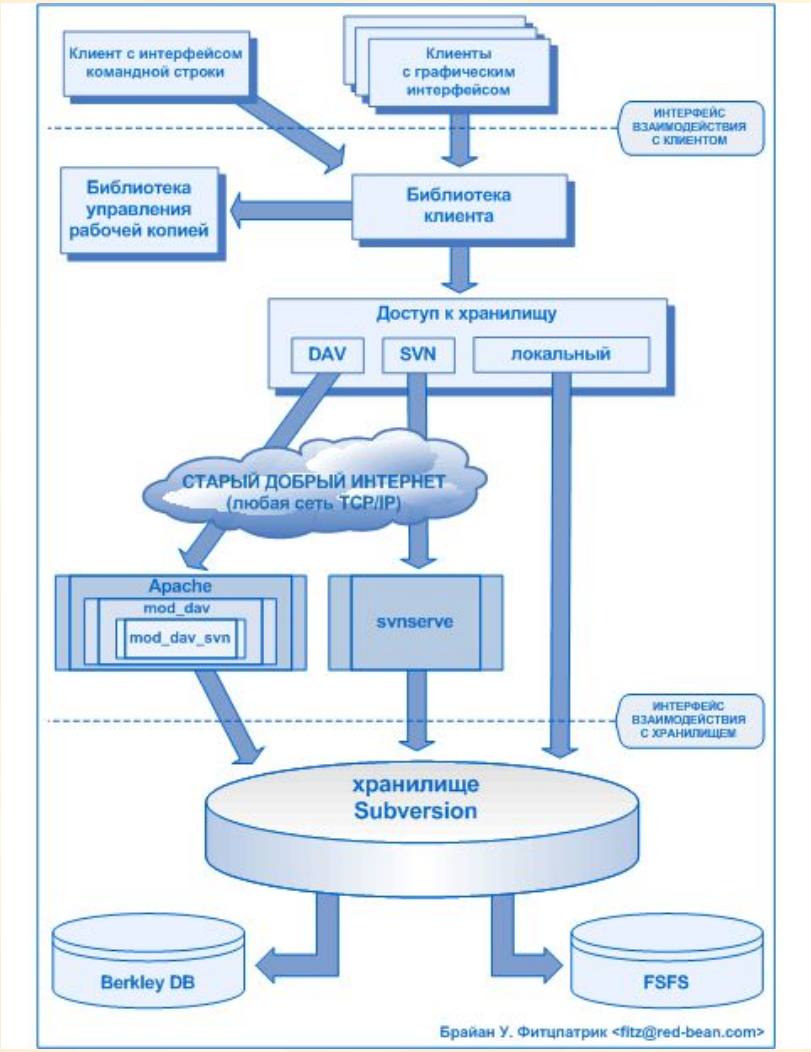
Правки

- Каждое новое состояние файловой системы хранилища называется *правкой*
- Каждая правка получает уникальный номер, на единицу больший номера предыдущей правки
- Начальная правка вновь созданного хранилища получает номер 0 и не содержит ничего, кроме пустого корневого каталога

Глобальность правок

- Номера правок в Subversion являются глобальными, т.е. относятся *ко всем*, а не только к отдельно взятым файлам
- Каждый номер правки соответствует целому дереву, отдельному состоянию хранилища после зафиксированного изменения

Архитектура Subversion

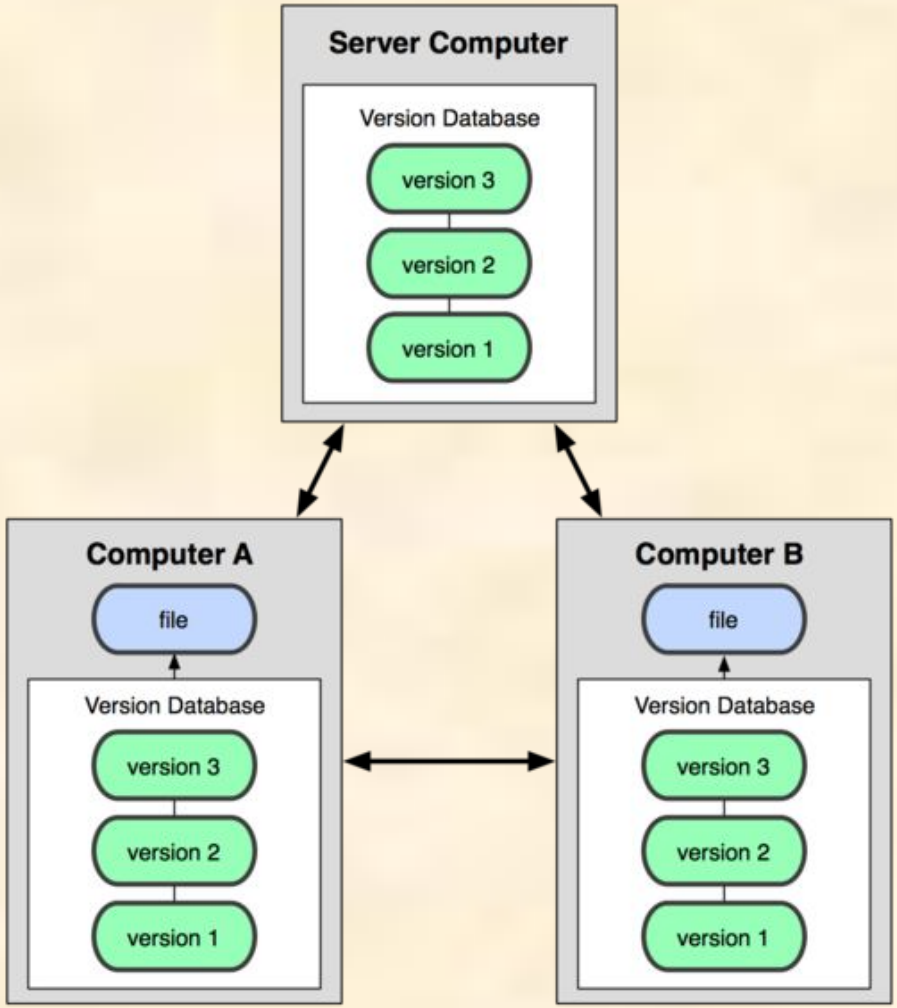


Недостатки Subversion

- Полная копия репозитория хранится на локальном компьютере в скрытых файлах, что требует достаточно большого объема памяти.
- Слабо поддерживаются операции слияния веток проекта.
- Не предусмотрено никаких штатных средств для полного удаления данных о файле из репозитория.

Распределенные системы контроля версий

- В распределенных системах контроля версий используются локальные репозитории клиентов
- Клиенты локально управляют версиями и могут обмениваться данными по сети



Распределенные системы контроля версий

- В условиях коллективной работы над проектом возможно создание централизованного репозитория, в который после проверки и синхронизации могут быть загружены данные из локальных репозиториев

Система Git

- Примером распределенной системы контроля версий является разработанная Линусом Торвальдсом система Git (2005 год)
- У каждого разработчика, использующего Git, есть свой локальный репозиторий, позволяющий локально управлять версиями. Затем, сохраненными в локальный репозиторий данными, можно обмениваться с другими пользователями.

Система Git

- Часто при работе с Git создают центральный репозиторий, с которым остальные разработчики синхронизируются (примером организации системы с центральным репозиторием является проект разработки ядра Linux’а)
- В этом случае все участники проекта ведут свои локальны разработки и скачивают обновления из центрального репозитория.

Система Git

- Когда необходимые работы отдельными участниками проекта выполнены и отлажены, они, после удостоверения владельцем центрального репозитория в корректности и актуальности проделанной работы, загружают свои изменения в центральный репозиторий

Система Git

- Наличие локальных репозиториев значительно повышает надежность хранения данных, так как при выходе из строя одного из репозиториев данные могут быть легко восстановлены из других репозиториев.
- Работа над версиями проекта в Git может вестись в нескольких ветках, которые затем могут полностью или частично объединяться, уничтожаться, откатываться и разрастаться во все новые и новые ветки проекта

Состояния файлов

- Git имеет три основных состояния, в которых могут находиться файлы:
  - зафиксированном (committed),
  - изменённом (modified),
  - подготовленном (staged).

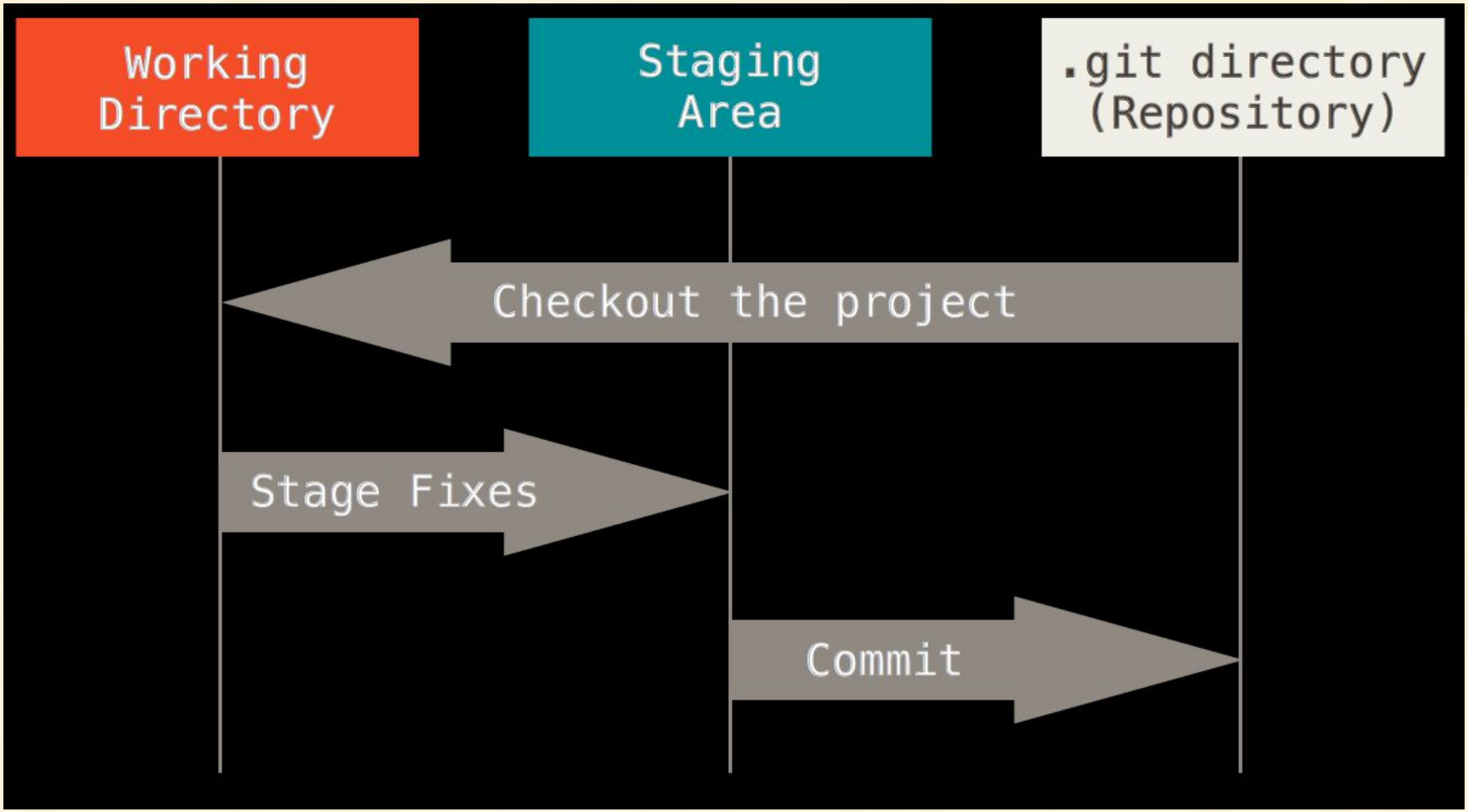
Состояния файлов

- Зафиксированный файл – это файл уже сохранён в локальной базе
- К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы
- Подготовленные файлы — это изменённые файлы, отмеченные для включения в следующий коммит

Секции Git

- Три основных секции проекта Git:
  - Git директория (Git directory),
  - рабочая директория (working directory)
  - область подготовленных файлов (staging area).

Секции Git


Working Directory
Staging Area
.git directory (Repository)
Checkout the project
Stage Fixes
Commit

Git директория

- Git директория — это то место где Git хранит метаданные и базу объектов вашего проекта.
- Это самая важная часть Git, и это та часть, которая копируется при клонировании репозитория с другого компьютера.

Рабочая директория

- Рабочая директория является снимком версии проекта.
- Файлы распаковываются из сжатой базы данных в Git директории и располагаются на диске, для того, чтобы их можно было изменять и использовать.

Область подготовленных файлов

- Область подготовленных файлов — это файл, располагающийся в Git директории, в нём содержится информация о том, какие изменения попадут в следующий коммит.
- Эту область ещё называют “индекс”, однако называть её stage область также общепринято.

Основные действия

- Базовый подход в работе с Git выглядит так:
  - Вы изменяете файлы в вашей рабочей директории
  - Вы добавляете файлы в индекс, добавляя тем самым их снимки в область подготовленных файлов
  - Когда вы делаете коммит, используются файлы из индекса, как есть и этот снимок сохраняется в вашу Git директорию
- https://git-scm.com/book/en/v2

Конец лекции