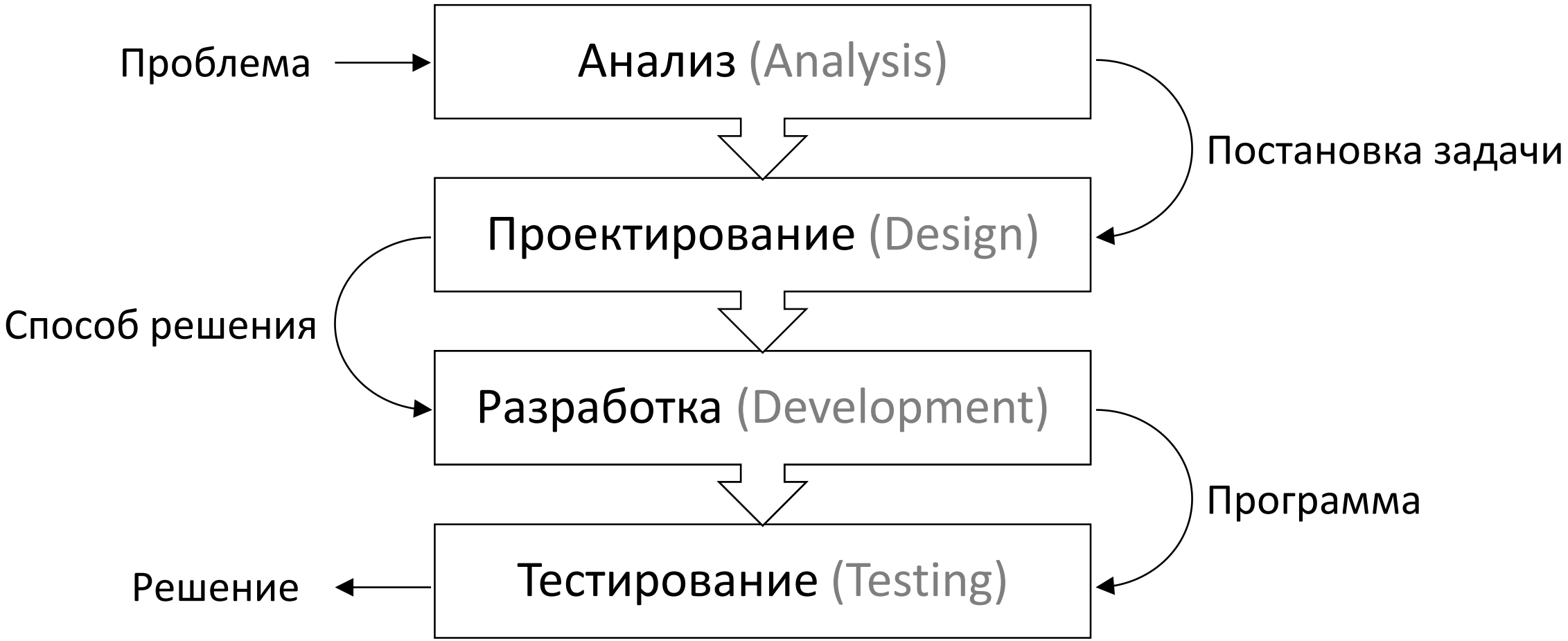
Средства автоматизации разработки программ

Лекция № 8

Курс «Технология программирования»

Процесс создания ПО



Элементы разработки

- Исходный код
  - Изменяется со временем
  - Его качество и надежность анализируемы
- Программа
- Документация
  - Должна соответствовать исходному коду

Часть 1

Статическое тестирование (static code analysis)

Статическое тестирование

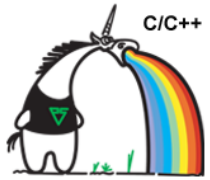
- Поиск ошибок в коде без запуска программы
- Прохождение не гарантирует правильность работы
- Ошибки бывают потенциальные:
  - Всегда истинные и всегда ложные условия, дублирование условий
  - Отсутствие обработки ошибок
  - Неправильная работа со стандартными функциями
  - Использование устаревшей функциональности
  - Нарушение правил кодирования

Пример: потенциально ошибочный код

```
unsigned short a, b, c;

for (size_t i = N; i >= 0; --i)
{
      std::cin >> a >> b >> c;
      if (a >= 0 && a < 256 &&
          b >= 0 && b < 256 &&
          c >= 0 && a < 256)
      {
            printf("%f", a*256*256 + b*256 + c);
      }
}
```

Популярные статические анализаторы кода

	PVS-Studio 	CppCheck
Операционная система	Windows	любая
Стоимость	5—10 тыс. €	бесплатно
Интеграция со средами разработки	Microsoft Visual Studio, Embarcadero C++ Builder	Code::Blocks, CodeLite, Eclipse, средства сборки Jenkins и Maven
Выдающиеся преимущества	богатство методов анализа	простота интеграции в процесс разработки любого проекта

Часть 2

Анализаторы производительности (performance analysis tools, profilers)

Анализ производительности

- Определяет время выполнения частей программы
- Позволяет выявить в программе «узкие места» (bottlenecks)
- Выполняется прогоном программы с замерами времени
  - Разрешение таймера компьютера важно
- Результат — случайная величина:
  - Комплекс условий:
    - Настройки оптимизации компилятора
    - Состояние окружения (загрузка процессора, сети, памяти, диска)
  - Для повышения точности оценки код должен выполняться много раз

Популярные анализаторы производительности общего назначения (для ЦП)

	GNU Profiler (GProf)	**Intel VTune Amplifier**
Операционная система	любая	Windows, Linux
Центральный процессор	Любой	совместимый с Intel
Стоимость	бесплатный	от 900 до 3100 $
Особенности	• Универсальность • Учет особенностей анализа производительности в *nix • Терминальный режим для встраиваемых систем	• Богатство инструментов • Учет особенностей процессоров Intel • Развитой интерфейс

Часть 3

Генераторы документации (documentation generators)

Кому, когда и зачем нужна документация?

- Разработчикам
  - Перечень сущностей (функции, типы данных)
  - Каким *в точности* заявлено поведение функций?
  - Обзор и анализ структуры кода
- Руководителям в ИТ
  - Как часть отчетности по проекту
  - Принятие решения об использовании библиотеки
    - Определение круга задач, решаемых библиотекой
    - Оценка стоимости внедрения
      - Объем и сложность структуры библиотеки
      - Наличие и качество документации *(замкнутый круг)*

Требования к документации

- Актуальность
  - Простота её поддержания
- Доступность
  - Время создания
  - Понятность языка для читателя
  - Формат документов
- Удобство работы
  - Алфавитный и предметный указатель
  - Перекрестные ссылки
- Оформление
  - Логичное
  - Эргономичное
  - Стандартное
  - Иллюстрации и схемы
    - Диаграмма классов
    - Зависимости модулей

Классические подходы

Чтение исходных текстов

- Самая актуальная информация
- Комментарии уже имеются
- Нужна квалификация
  - Сложно для руководителя
  - Надолго отвлекает разработчика
- Структура диктуется техническими соображениями, часто неудобна
  - Трудно получить общее представление
- Выразительного оформления нет

Написание документации вручную

- Логичная структура
- Перекрестные ссылки, оглавление, поиск, иллюстрации
- Эргономичное оформление
- Необходимость актуализации вручную при изменениях кода
- Высокая трудоемкость
  - Нужен технический писатель

Решение — генераторы документации

- По исходному коду можно сформировать документацию
  - Оглавление, алфавитный указатель, поиск
  - Диаграммы (структура библиотеки)
- В коде есть комментарии
  - Их можно включить в документацию
  - В них можно указывать команды генератору документации
    - Управление форматированием
    - Перекрестные ссылки
      - Частично генерируются автоматически
    - Разнесение сущностей по логическим разделам

Свойства генерируемой документации

- По качеству структуры и оформления — как рукописная
  - Несколько меньшая гибкость
  - Приходится вносить много информации в комментарии
    - Код становится труднее читать
    - IDE может формировать контекстные подсказки по ним
- Актуализируется автоматически
- Малая трудоемкость
  - Достаточно поддерживать комментарии в коде
  - Желательно соблюдать формат комментариев
  - Можно формировать документы, файлы справки, версии для web
- Привычный формат

Популярные генераторы документации

- Doxygen (C-подобные языки)
  - DoxyBlocks — расширение Code::Blocks

  Индустриальный стандарт для C++
- JavaDoc (Java)

  Индустриальный стандарт для Java
  - PhpDocumentor (PHP), JSDoc (Java Script)
- Sphinx (Python и другие)
  - Python имеет встроенные в язык `"""docstrings"""`
- Doc-O-Matic

Пример: документирующий комментарий

```
/** @brief Производит снятие средств со счета.
  *
  * Снятие средств со счета возможно, если сумма к снятию положительна
  * и не превышает баланса на счете, а сам счет не заморожен
  * (@see isFrozen). После успешного снятия сумма средств на счете
  * (@see getBalance) уменьшается на величину @paramref amount.
  *
  * @param amount Сумма средств к снятию.
  * @returns Удалось ли произвести снятие (@code true),
  *          или возникла ошибка (@code false).
  */
bool withdraw(double amount);
```

Часть 4

Системы контроля версий (Version Control Systems, VCS)

Задачи системы контроля версий

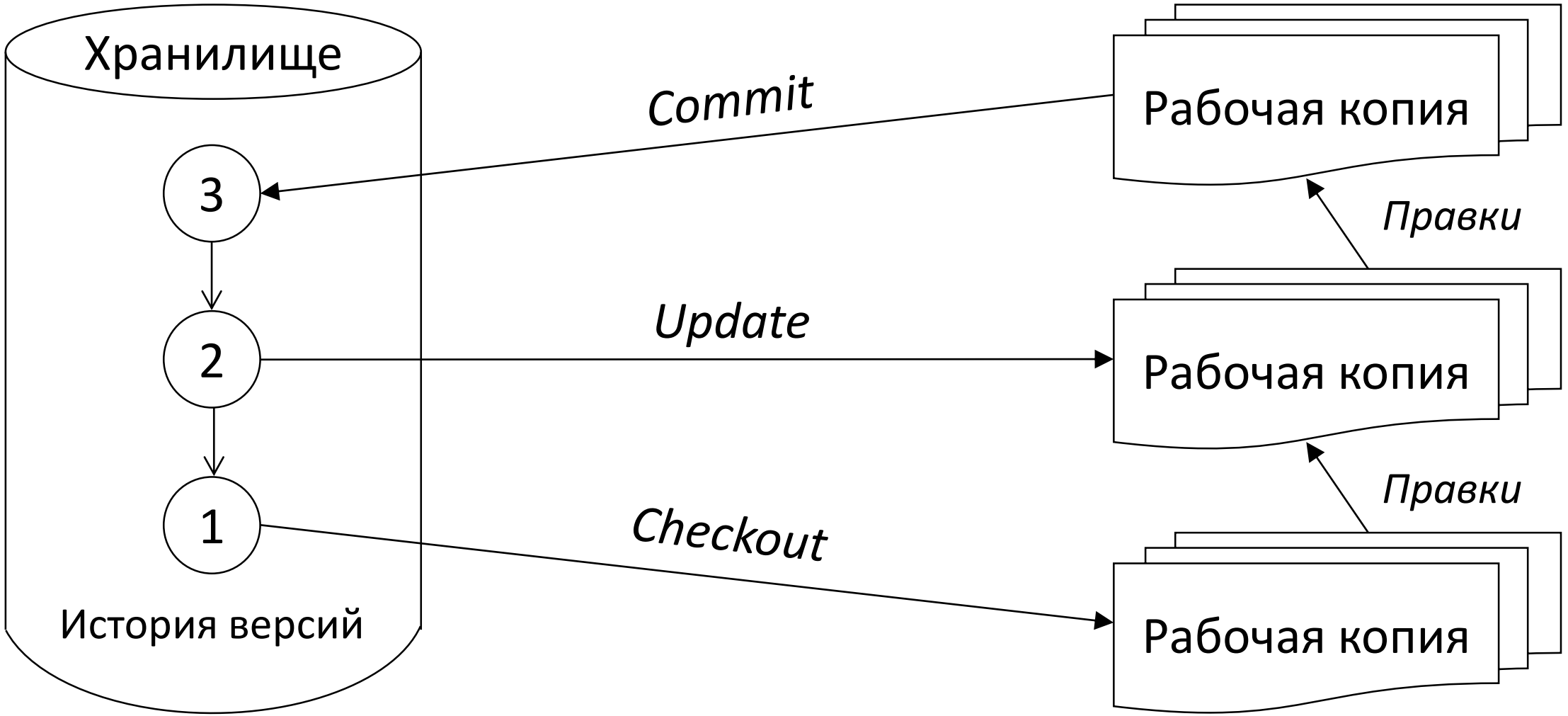
- Хранить сообща различные версии проекта
- Получать предыдущие версии файлов проекта
- Хранить комментарии к вносимым изменениям
- Отслеживать когда, какие, кем и зачем были внесены изменения
- Вести параллельную разработку разных частей проекта
- Получать изменения между двумя произвольными версиями
- Облегчить задачу поиска мест возникновения логических ошибок

Цит. по презентации «Subversion и разработка web-приложений», Фетисов Н. А. и Фетисов Ф. А., 2013

Основные понятия VCS

- **Хранилище (repository),** или репозиторий, —
  место хранения файлов и их версий, служебной информации.
- **Версия (revision),** или ревизия, —
  состояние всего хранилища или отдельных файлов в момент времени.
- **Commit** («вклад», не переводится) —
  процесс создания новой версии.
- **Check-out** («переход к…», переводится редко) —
  процесс получения рабочей копии версии из хранилища.
- **Рабочая копия (working copy) —**
  текущее состояние файлов проекта (любой версии), полученных из хранилища.
- **Обновление (update) —**
  обновление состояния рабочей копии из репозитория.

Схема работы с VCS

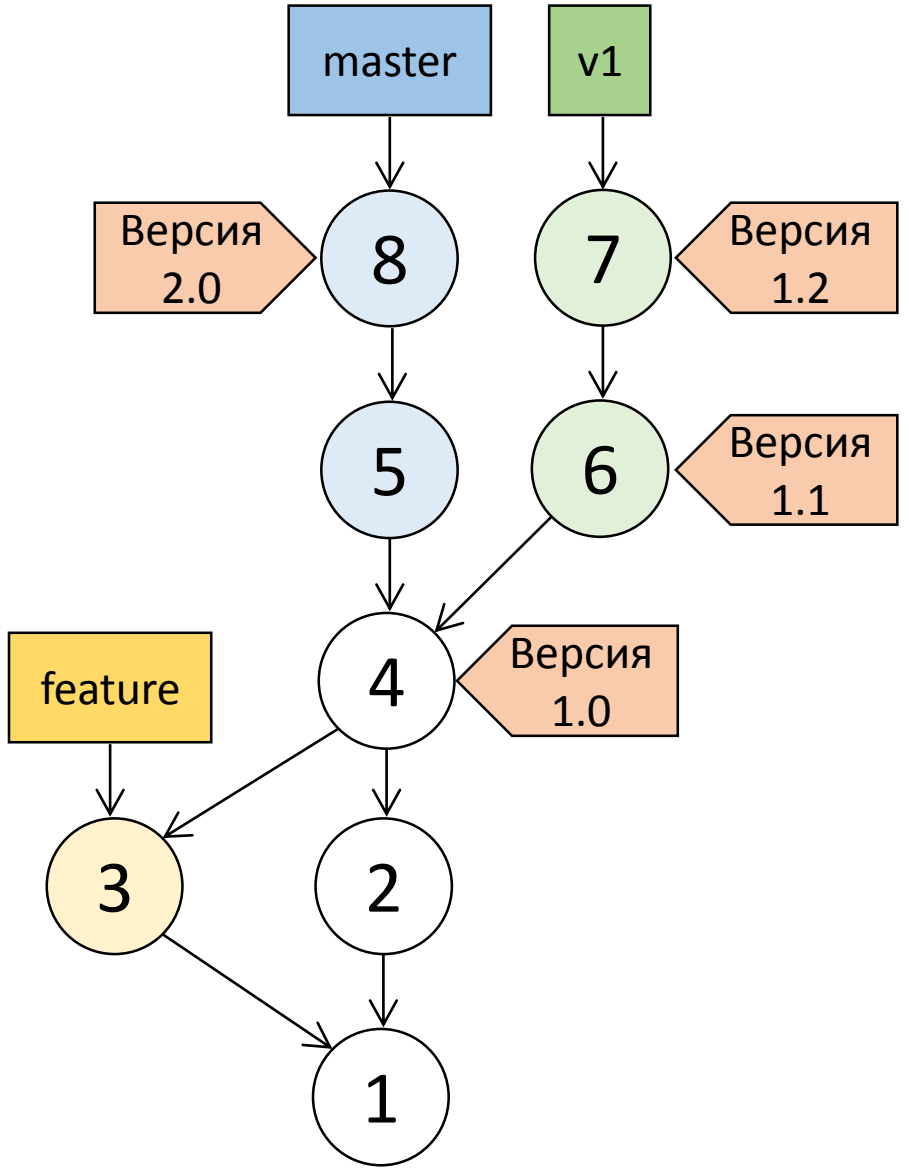


Основные понятия VCS (продолжение)

- **Слияние (merge) —**
  объединение независимых изменений в единую версию в хранилище.
- **Конфликт (conflict) —**
  ситуация, когда VCS не может автоматически применить внесённые изменения (т. е. когда параллельно были исправлены одни и те же места в файлах).
- **Разность (difference, diff) —**
  построчные различия между файлами (разных версий).
- **Заплатка (patch), патч —**
  файл-инструкция, какие правки нужно внести (по сути, это diff).

Ветки и метки

- **Ветка (branch) —** один из параллельных участков истории, исходящих из одной версии (точки ветвления)
  - Обычно есть главная ветка (master), или ствол (trunk)
  - Между ветками возможно слияние
- **Метка (tag) —** отмеченная версия



Что есть пункт истории?

	Версия (revision)	**Набор изменений (changeset)**
Атрибуты	Уникальный порядковый номер	• Уникальный идентификатор (хэш содержимого, не номер) • Ссылка на предыдущий набор
Содержимое	Состояние всего хранилища	Изменения файлов по отношению к любому предыдущему состоянию
Ветка — это…	каталог в составе проекта, у нее общая история с другими	…цепочка наборов изменений, имеющая общую точку с другими
Слияние — это…	большое сравнение двух состояний, разрешение конфликтов	…сравнение двух цепочек набор за набором, разрешение конфликтов изменений

Виды систем контроля версий

Централизованные

Единая история хранится
в центральном репозитории,
у пользователей — только рабочие копии.

- Простота использования
- Необходимость подключения к сети
- Удобство резервного копирования
- Удобство разделения прав доступа
- Имеется единственно верная версия истории
- Малая гибкость, сложность использования веток и слияний (revisions)

Распределенные

Каждый пользователь владеет полноценной
и равноправной копией хранилища
с независимой историей.

- Двухфазный commit: запись в локальную историю и пересылка изменений другим
- Подключение к сети к сети не требуется
- «Центральное» хранилище условно
- Синхронизация хранилищ не автоматическая
- Большая гибкость, удобство работы с ветками и слиянием (changesets)

Некоторые VCS и их особенности

- Subversion (SVN)
  - Одна из старейших, и потому все еще популярна
  - Централизованная
- Git
  - Распределенная, популяризовала этот тип
  - Самая распространенная на сегодня (GitHub, BitBucket)
- Mercurial (Hg)
  - Распределенная, похожа на Git
  - Широкие возможности по управлению хранилищами
- Perforce
  - Основана на Git, но требует центрального хранилища
  - Улучшенная работа с файлами не-кода
  - Ядро сложной системы ведения проекта
- Анонсированная VCS фирмы JetBrains
  - Оригинальное ядро
  - Требует центрального хранилища
  - Все commit-ы перемещаются в центральное хранилище, но автор может управлять их видимостью
  - Рабочая копия — часть хранилища
  - Интегрирована с продуктами JetBrains